

The Software Therapist: Usability Problem Diagnosis through Latent Semantic Analysis

STTR Phase II Final Report

14 June 2006

Report Documentation Page

Contract Number: FA9550-04-C-0057

Start Date: 15 June 2004

STTR Topic: AF03-T001

**Automated Diagnosis of Usability Problems Using Statistical
Computational Methods**

Participating Institutions:

- **Knowledge Analysis Technologies, LLC
(Pearson Knowledge Technologies)**
- **Virginia Polytechnic Institute and State University (Virginia Tech)**

Principal Investigator, Small Business Concern:

Randall Sparks, Ph.D.
Knowledge Analysis Technologies, DBA Pearson Knowledge Technologies
4940 Pearl East Circle
Boulder, CO 80301

Principal Investigator, Research Institution:

Rex Hartson, Ph.D., Prof. Emeritus
Virginia Tech
Blacksburg, VA 24061

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 14-09-2006		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 15 Jun 04 to 14 Jun 06	
4. TITLE AND SUBTITLE The Software Therapist Usability Problem Diagnosis through Latent Semantic Analysis				5a. CONTRACT NUMBER FA9550-04-C-0057	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Randall Sparks – Knowledge Analysis Technologies Rex Hartson, Jon Howarth – Virginia Tech				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Knowledge Analysis Technologies Virginia Polytechnic Institute DBA Pearson Knowledge Technologies 460 Turner Street, Suite 206 4940 Pearl East Circle Blacksburg, VA 24060 Boulder, CO 80301				8. PERFORMING ORGANIZATION REPORT	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) USAF, AFRL AF Office of Scientific Research 4015 Wilson Blvd. Room 713 Arlington, VA 22203-1954 AFOSR/NL				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-SR-AR-TR-06-0471	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release – Distribution A					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The work we report on here addresses the problem of low return on investment in software usability engineering and offers support for usability practitioners in identifying, understanding, documenting, and fixing usability problems. It does this by (1) validating and extending a structured knowledge framework of usability concepts for organizing and relating usability data to design flaws and solutions, (2) specifying a usability data management cycle to support a diagnosis process that is iteratively interleaved with data collection, and (3) developing a software system for usability engineering practitioners that includes components to support the activities of usability data collection, organization, and problem analysis and reporting, as well as automated support for usability problem diagnosis using a sophisticated statistical technique for the analysis of text. In this report, we briefly review the motivations and background for this work and describe the User Action Framework (UAF) and Latent Semantic Analysis (LSA); describe the design, development, and use of the Software Therapist system; discuss the research done on applying LSA to usability engineering; describe the usability content library collected as part of the project; and discuss the evaluation of the system as well as some lessons learned and possible future directions resulting from the project.					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF: U			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 102	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

Table of Contents

1	INTRODUCTION & OVERVIEW.....	15
1.1	TECHNICAL BACKGROUND.....	18
1.2	THE SOFTWARE THERAPIST TOOLS.....	18
2	SOFTWARE TOOL DEVELOPMENT AND USE.....	18
2.1	THE DATA COLLECTION, ANALYSIS, AND REPORTING TOOL.....	18
2.1.1	<i>Usability Problem Database.....</i>	<i>18</i>
2.1.1.1	Context.....	18
2.1.1.2	Format.....	18
2.1.2	<i>Usability Problem Diagnosis.....</i>	<i>18</i>
2.1.2.1	Full Diagnosis.....	18
2.1.2.2	Partial Diagnosis.....	18
2.1.2.3	Immediate Intention.....	18
2.1.2.4	Micro-Iteration.....	18
2.1.2.5	Diagnosis Wizard.....	18
2.1.3	<i>Technical Specifications.....</i>	<i>18</i>
2.1.3.1	Support for Context.....	18
2.1.3.2	Support for Format.....	18
2.1.3.3	Support for Diagnosis.....	18
2.1.3.3.1	Support for Full Diagnosis.....	18
2.1.3.3.2	Support for Partial Diagnosis.....	18
2.1.3.4	Usability Problem Inspection.....	18
2.2	THE SOFTWARE THERAPIST BROWSER.....	18
2.2.1	<i>Purpose & Design of the ST Browser.....</i>	<i>18</i>
2.2.2	<i>Using the Software Therapist Browser with the UAF.....</i>	<i>18</i>
2.2.3	<i>Usability Problem Diagnosis in the ST Browser.....</i>	<i>18</i>
2.2.3.1	LSA and Semantic Spaces for Usability Analysis.....	18
2.2.3.2	Using LSA for Problem Diagnosis.....	18
2.2.3.3	Using LSA to search for related literature.....	18
3	RESEARCH ON THE APPLICATION OF LSA TO USABILITY ENGINEERING.....	18
3.1	VALIDATION OF UAF CONTENT & STRUCTURE.....	18
3.2	ROLE OF “DISTINGUISHERS” IN PROBLEM DIAGNOSIS.....	18
3.3	HUMAN VALIDATION OF LSA CLASSIFICATION.....	18
4	USABILITY CONTENT LIBRARY COLLECTION.....	18
4.1	USABILITY PROBLEM REPORT LIBRARY.....	18
4.2	UAF TERMINAL NODE SERVICES.....	18
4.2.1	<i>Case Studies.....</i>	<i>18</i>
5	EVALUATION.....	18
5.1	EXPLORATORY STUDIES.....	18
5.2	ANALOGY TO MEDICAL DIAGNOSIS.....	18
5.3	FORMATIVE STUDIES OF THE WIZARD.....	18
5.4	FORMATIVE EVALUATION.....	18
5.4.1	<i>Low-Fidelity DCART Prototype.....</i>	<i>18</i>
5.4.2	<i>Formative Evaluation of the ST Browser.....</i>	<i>18</i>
5.5	FIELD TRIALS.....	18
5.5.1	<i>Field Test of High-Fidelity Prototype Version.....</i>	<i>18</i>

5.5.1.1	Typical run-of-the-mill usability problems	18
5.5.1.2	Positive feedback	18
5.5.1.3	Constraints	18
5.5.1.4	Complexity.....	18
5.5.1.5	Work flow model rather than data structure model.....	18
5.5.1.6	Expert vs. Novice Users.....	18
6	CONCLUSION	18
7	REFERENCES.....	18
8	APPENDIX A. LATENT SEMANTIC ANALYSIS.....	18
9	APPENDIX B. THE USER ACTION FRAMEWORK.....	18
9.1	THE INTERACTION CYCLE	18
9.2	THE UAF: ADDING A STRUCTURED USABILITY KNOWLEDGE BASE	18
9.3	RELATED WORK	18
9.3.1	<i>Model-based frameworks.....</i>	<i>18</i>
9.3.2	<i>Classifying usability problems by type</i>	<i>18</i>
10	APPENDIX C. REVIEW OF PROBLEMS ADDRESSED.....	18
10.1	LOW RETURN ON INVESTMENT DOWNSTREAM FROM USABILITY TESTING	18
10.2	NEED FOR THEORY-BASED CONCEPTUAL STRUCTURE	18
10.3	NEED FOR INTEGRATED SUITE OF USABILITY ENGINEERING TOOLS	18
10.3.1	<i>Need for theory-based usability problem analysis, diagnosis, and reporting tool.....</i>	<i>18</i>
10.3.1.1	Need for integrated usability data management tool	18
10.3.1.2	Need for systematic, theory-driven usability inspection tool	18
10.3.1.3	Need for usability design tool to guide interaction design and redesign activities.....	18
10.3.1.4	Need for extensibility to new interaction styles	18
10.3.1.5	Need for intelligent analysis systems to match solutions to usability problems.....	18
11	APPENDIX D. BENCHMARK TASKS AND USABILITY PROBLEMS FROM THE FORMATIVE EVALUATION OF THE DCART LOW-FIDELITY PROTOTYPE	18
11.1	EXAMPLE BENCHMARK TASKS	18
11.2	USABILITY PROBLEMS	18
12	APPENDIX E. EXPLORATORY TASKS FOR PARTICIPANTS IN THE FIELD TEST OF THE DCART HIGH-FIDELITY PROTOTYPE	18
	TABLE OF FIGURES	18

1 Introduction & Overview

Increasing the usability of software by including usability engineering in the development process has become common practice. In fact, usability is now one of the primary factors in determining whether a software application succeeds or fails to achieve its goals in the competitive software marketplace or in the corporate or military information technology environment. However, usability engineering efforts have been focused largely on usability problem data capture through usability testing and usability inspection. Downstream in the process from usability testing, developers often receive a low return on their investment in usability engineering because of information losses. The causes of these losses can be attributed to a lack of an adequate conceptual framework for organizing usability data and a lack of an appropriate usability data management cycle to support problem diagnosis and analysis (Gray & Salzman 1998). Without an adequate conceptual framework, the analysis process is likely to be ad hoc, and cannot use a standard vocabulary. In addition, the sequential nature of collection and analysis activities in the existing usability data management cycle make it difficult for practitioners to collect the right information and record it in an appropriate form. The data collected often fail to support the successful transformation of raw usability data into effective inputs to redesign for fixing the problems found during testing. These problems can be exacerbated by inadequate levels of training or experience by usability practitioners.

The work we report on here addresses the problem of low return on usability investment and offers support for practitioners in identifying, understanding, documenting, and fixing usability problems. It does this by (1) validating and extending a structured knowledge framework of usability concepts for organizing and relating usability data to design flaws and solutions, (2) specifying a usability data management cycle to support a diagnosis process that is iteratively interleaved with data collection, and (3) developing a software system for usability engineering practitioners that includes components to support the activities of usability data collection, organization, and problem analysis and reporting, as well as automated support for usability problem diagnosis using a sophisticated statistical technique for the analysis of text. The problems addressed by this work were discussed in more detail in the proposal and are summarized in Appendix C (section 10).

In this report, we will briefly review the motivations and background for this work and describe the User Action Framework and Latent Semantic Analysis (this section); describe the design, development, and use of the Software Therapist system (section 2); discuss the research done on applying LSA to usability engineering (section 3); describe the usability content library collected as part of the project (section 4); and discuss the evaluation of the system as well as some lessons learned and possible future directions resulting from the project (sections 5 and 6).

1.1 Technical Background

The work reported here was guided by the two technologies around which the project proposal was built: Latent Semantic Analysis and the User Action Framework. Latent Semantic Analysis (LSA) is a statistical technique for representing and comparing the meaning of words and texts. It has been developed and applied successfully to a variety of applications for over 20 years. For this project, we have applied LSA in several ways, in each case as a type of search technique, to

identify documents¹ that are semantically related to one another in useful ways. A brief summary of LSA is provided in Appendix A (section 8).

The User Action Framework (UAF) is a knowledge base of usability concepts, issues, and problems, structured hierarchically to facilitate the classification and analysis of usability problems and the discovery of appropriate and effective solutions to these problems. For this project, the UAF was validated, extended with new material, and embodied in the Software Therapist system to make it more accessible and useful to usability practitioners. A summary of the UAF is provided in Appendix B (section 9).

1.2 The Software Therapist Tools

The Software Therapist system consists of two primary components: the Data Collection, Analysis, and Reporting Tool (DCART) and the Software Therapist (ST) Browser. Together, these components constitute a usability engineering problem solving environment based on the UAF and LSA. The DCART component of the system guides and assists usability practitioners² in the tasks of setting up usability trials and walkthroughs; and collecting, storing, and analyzing the data from these activities. The user's first steps in DCART involve defining the organizations, projects, tasks, participants, usability specifications, goals, etc. that are involved in usability analysis. This information is stored automatically in a database. Figure 1 shows a screenshot of DCART with its main Workspace View being used to edit the definition of an example project. The design and use of DCART is described in greater detail in Section 2.1.

After this contextual information has been defined, the usability trials³ can be conducted, during which DCART can be used to record data, primarily “critical incidents” and other data relevant to the generation of a *usability problem report*. These reports can then be used to diagnose and further analyze the problems that occurred during the trial and may lead to a design change that can effectively fix the problem or otherwise successfully address the issue. Problem reports are stored in a database and may be edited from DCART at any time.

¹ In this report, we will follow the common practice in the literature on LSA (and, more broadly, of information retrieval and related fields) of using the term *document* to refer to any text that can be the object of a search and/or retrieval operation. Most important to note here is that a document in this sense is most commonly and usefully defined to be of roughly paragraph size. For example, it is often of more benefit to a user when the result of a search is a set of relevant paragraphs (or, perhaps pages or similar passages), rather than entire articles or books. However, we may also refer at times to these larger “documents”, for example the articles, papers, and books that constitute the source documents for building semantic spaces (for which, see below).

² In this report, we will refer to users of the Software Therapist system variously as “users”, “usability engineers”, or “(usability) practitioners”. A usability practitioner could be anyone engaging in the process of usability engineering, whether a student, professional, or academic in human-computer interaction or related field, or a software engineer or designer of any type of system with a human interface.

³ The Software Therapist can be used with *usability trials*, both those in which real or surrogate users of the system under study interact with the system (or prototype) to perform the kinds of tasks the system is intended for, as well as in *usability walkthroughs*, in which usability practitioners and/or users “walk through” the functions of the target system—typically using a surrogate, such as a prototype or even paper—and record impressions and other observations. See Nielsen (1989), Nielsen & Mack (1994). In this report, we will usually use “usability trial” to refer to any of these techniques, unless a finer distinction is necessary.

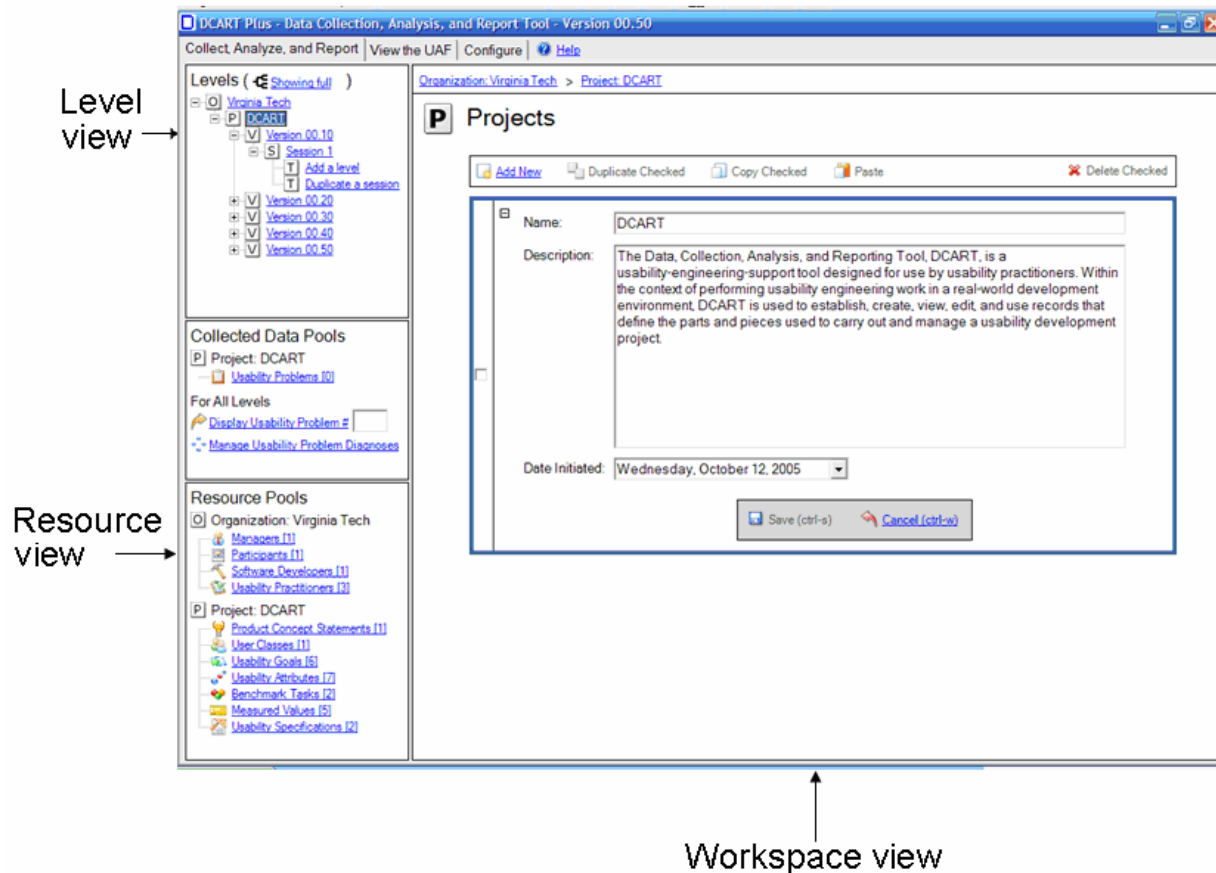


Figure 1: Screenshot of the DCART component of the Software Therapist system.

After problem reports have been generated, the particular problem at issue in each needs to be properly diagnosed before an appropriate solution can be determined. This is where the UAF comes into play. By starting at the root of the UAF hierarchy and selecting the appropriate child node at each branching point, the usability practitioner can arrive at a path through the UAF that constitutes a full diagnosis of that problem (and, thus, a basis for a solution). Prior to the work reported here, this process had been demonstrated to be an effective technique (Andre et al. 2001, 2000, 1999), but had to be carried out entirely manually—that is, without any guidance that was sensitive to the particular problem report at hand. Due to the complexity of this type of diagnosis task, the size of the UAF, and other complicating factors (such as incompleteness of the problem report and uncertainty of the trial user's intent), this technique was effective primarily in the hands of usability experts with significant prior experience using the UAF. The work reported here describes techniques and the implementation of tools that make the strengths of using the UAF for this type of analysis more readily available to others (e.g., to usability experts without familiarity with the UAF, less-experienced usability practitioners, software developers, and students). These tools are also designed to help achieve more consistent and expedited use by experts and non-experts alike.

The tools for UAF-based usability problem diagnosis include the Diagnosis “Wizard” and the Software Therapist (ST) Browser. From a problem description in DCART, users may invoke the Diagnosis Wizard, which breaks the process of finding a diagnosis path through the UAF into a

sequence of binary decisions in the form of either/or answers to relatively simple questions. Alternatively, users may use the ST Browser, which provides a variety of LSA-based search features to assist the user in determining an appropriate diagnosis and finding a solution for usability problems.

Figure 2 shows a screenshot of the ST Browser, opened to the UAF. The top left panel shows a Table of Contents (TOC) view of the UAF in the selected *Contents* tab. When browsing the UAF, each item in the TOC view represents one node in the UAF. In the TOC view, the user may expand or collapse selected nodes to browse to any part of the UAF. The node selected by the user in Figure 2 is the root node of the UAF, so that its contents are displayed in the main document display panel on the right. This shows the node's content: the title ("User Action Framework"), a description of the node, examples, etc.

The ST Browser also includes several controls for navigation and search of the content displayed and other, related content. Some of these are similar to common web browser controls, such as the History buttons in the top middle of the display. Others are more specialized, as they implement the LSA-supported search features of the Software Therapist system. One of these is the Search Panel at the bottom left of the browser. The controls here enable users to retrieve usability problem reports from a database, display and edit them here,⁴ and invoke one of several LSA-supported search strategies, such as to find candidate UAF diagnoses for the problem report, find passages of HCI literature that may be related to the problem in the report, find related problem reports, etc. The ST Browser, its function and use are described in more detail in Section 2.2.

⁴ The text of a problem report can be edited here to modify the search to be conducted. To permanently modify the database record of a problem report, the Usability Problem Review Form of DCART is used. (See section 2.1.3.2.)

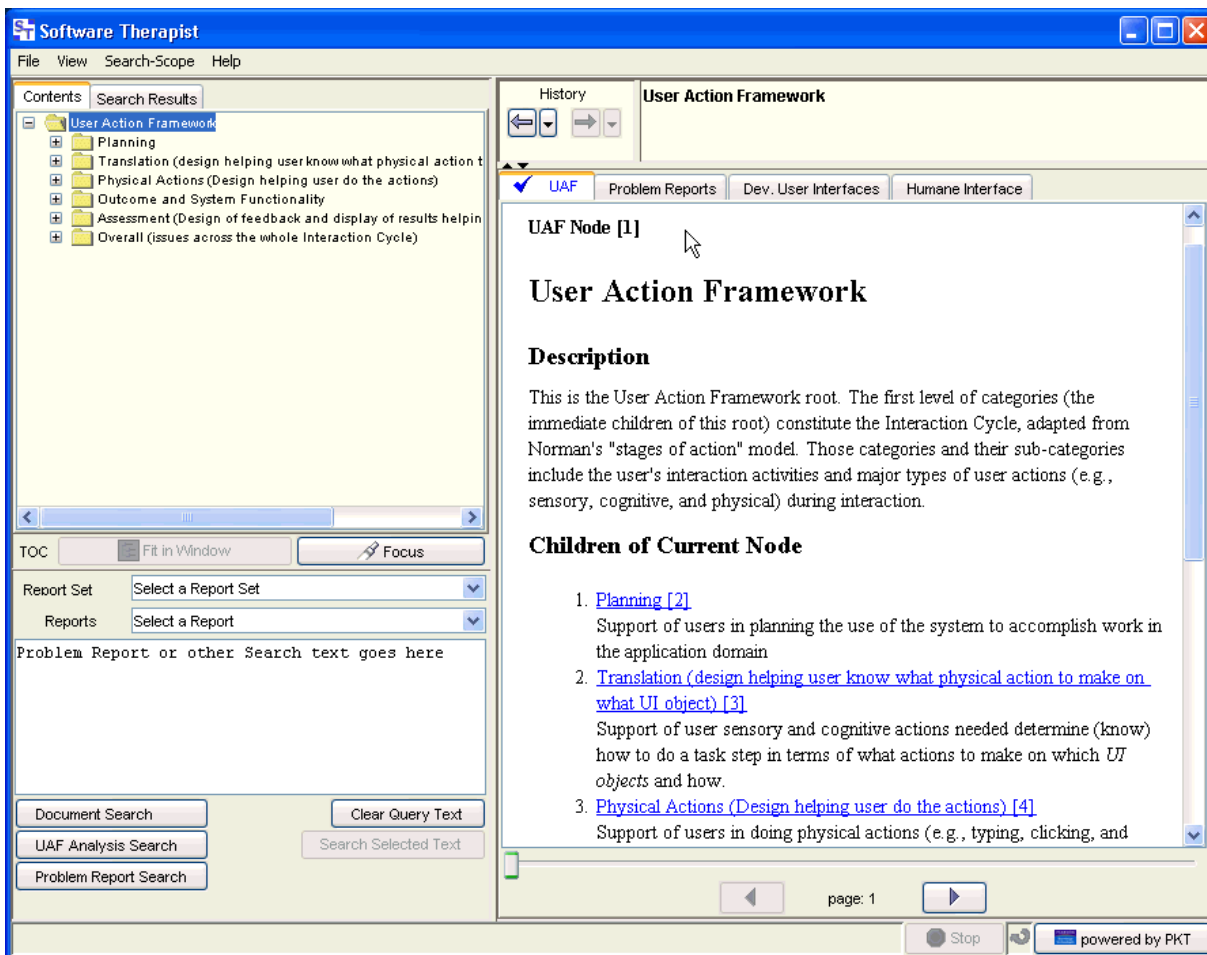


Figure 2. The ST Browser opened to the UAF, showing the contents of the root node of the UAF.

2 Software Tool Development and Use

The original phase II work plan describes three separate tools, the Problem Reporting Tool, the Usability Problem Inspector, and the Usability Design Guidelines Tool. Through the iterative design and implementation process we followed, these were merged into two primary components: the Data Collection, Analysis, and Reporting Tool (DCART) and the Software Therapist (ST) Browser, which together constitute the Software Therapist system.

2.1 The Data Collection, Analysis, and Reporting Tool

2.1.1 Usability Problem Database

The design of the Software Therapist system and, in particular, its DCART component, was guided by several key concepts that we developed and evolved over the course of our work. The process of usability problem analysis is driven to a large extent by usability problem reports. To support this process in a tool requires a repository of problem reports, which we have

implemented in the form of a usability database. (See section 2.1.1.) The usability database is populated by usability problem records, which consist of two parts: a *context* and a *format*.

2.1.1.1 Context

Associating usability problem records with a particular context reduces the amount of data that a practitioner must record to specify a usability problem. Context implies an understanding of the circumstances in which something occurs.

We have defined a number of levels of context to create a hierarchical context inside of which usability problem records are nested. The top level of the hierarchy involves the broadest context. The second level of the hierarchy is nested inside the first and has a narrower context. Each progressive level is nested inside the previous one and has a narrower context. As a result, the more deeply nested the level, the more specific the context.

Figure 3 shows the six levels of hierarchical context that we have developed in an attempt to help practitioners better specify and capture context: organization, project, version, session, task run, and problem. The organization, project, and version levels provide general application context, such as the need or purpose for the application and its target environment. Finally the problem context, the most deeply nested level, contains details about usability problems experienced by participants.

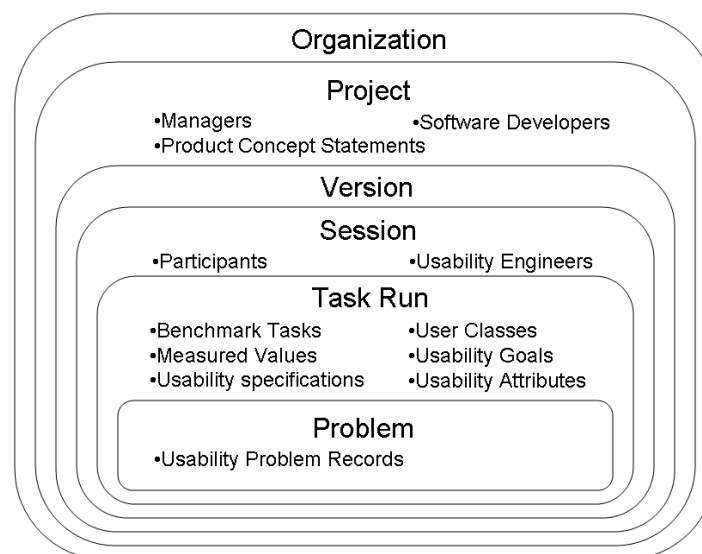


Figure 3: Levels of hierarchical context and associated resources in DCART.

The organization level contains details about an organization. The project level contains details about software applications that an organization wants to evaluate, and the version level contains details about each of the versions of a project. The session level represents a session between one or more facilitators and one or more participants. The task run level represents one task as performed by a participant or participants as part of a session. Finally, the problem level represents a usability problem experienced by a participant during a task run.

All levels of the hierarchy except the organization and version levels have resources associated with them. The term resource is used for people or objects that perform a function at a given context level. The following is a list of resources by context level:

- **Project**
 - **Managers** – Manage projects by assigning individuals to them and allocating resources for them.
 - **Software developers** – Develop prototypes for use in the usability evaluation sub-process.
 - **Product concept statements** – A brief descriptive summary of the product being developed. As a kind of mission statement for the project, the product concept statement is typically 50-75 words in length and sets the focus and scope for the design team in the overall development effort.
- **Session**
 - **Participants** – Participate in usability evaluation sessions.
 - **Usability practitioners** – Collect, analyze, and report data in the usability evaluation sub-process.
- **Task Run**
 - **User classes** – Descriptions about the various roles users play while interacting with the system. These descriptions provide a set of attributes such as users' knowledge of computers or users' training and application-related experiences and guide the overall design effort. For example, for a user class with little to no computer knowledge or training, the system design will probably include a significant amount of "handholding" with detailed instructions for each stage of the interaction. This might contrast with the design for another user class with extensive computer knowledge and domain expertise where the focus will probably be on providing "power" features with shortcut keys.
 - **Usability goals** – High-level objectives stated in terms of usability and design of user interaction. They reflect real use of a product in the real world and determine what is important to an organization and its users. Usability goals may be market driven. Examples include customer satisfaction and walk-up-and-use usability.
 - **Usability attributes** – The general usability characteristic that is to be measured for an interface. Some common usability attributes include: initial performance, long-term performance, learnability, retainability, advanced feature usage, first impression, and long-term user satisfaction.

- **Benchmark tasks** – Standardized unambiguous descriptions of representative, frequently performed, and critical tasks, to be used in usability evaluation tests.
- **Measured values** – Quantitative data that are collected from a user during or after a user interacts with a software system. These values can be either objective or subjective. Objective measured values are quantitative measures of observable user performance while performing tasks with a user interface. Subjective measured values are quantitative measures based on user opinion about the user interface.
- **Usability specifications** - Quantitative usability goals against which user interaction design is measured. They include target levels for usability attributes and are often used as a guide and process management tool to know whether the development process is converging toward a successful design.
- **Problem**
 - **Usability problem record** – A record of a user experiencing a usability problem. The fields included in a usability problem record are described in Section 2.1.3.2.

2.1.1.2 Format

In addition to context, a consistent usability problem report format for usability problem records would standardize the way in which usability problem data are recorded. Such a format would make facilitators aware of needed usability data in the usability data collection stage and provide problem analysts with more consistent data in the usability problem analysis stage. Usability problem records exist within the problem context (Figure 3).

Based on our synthesis of the related work and our own experience, we suggest the use of a report format that includes the following three types of data and associated fields:

- **Descriptive** – These data describe the usability problem itself including outcomes experienced by the participant.
 - Name of the usability problem
 - Description of the usability problem
 - Screenshot of the usability problem
 - The interface object(s) involved
 - Relevant designer knowledge
- **Diagnostic** – These data describe the cause of the usability problem.
 - A usability problem diagnosis

- **Prescriptive** – These data contain suggestions for fixing the usability problem.
 - Suggestions for fixing the usability problem
 - Estimate of the cost of fixing the usability problem
 - Estimate of the severity of the usability problem

2.1.2 Usability Problem Diagnosis

To support practitioners in the process of usability problem diagnosis, we needed to develop a practical approach to diagnosing usability problems with the UAF. The process of diagnosis with the UAF involves associating a usability problem with a path of UAF nodes that completely describes the usability problem and its causes. We cover two levels of diagnosis: full and partial, both of which are supported by the Software Therapist system.

2.1.2.1 Full Diagnosis

The process of full diagnosis with the UAF involves associating a particular usability problem with a path of UAF nodes that completely describes the usability problem and its causes. Figure 4 shows the Interaction Cycle (Section 9.1) extending into the full UAF, a tree structure of usability concepts representing the multidimensional space of design features and usability problem data. The tree continues many levels deeper to the right; only three levels are shown in the illustration. Each level of the tree structure maps to a dimension, and each node (diagnosis choice) at a given level maps to an attribute or value within that dimension. Selecting one of the nodes at a given level is equivalent to removing attributes that don't apply to a given usability situation, thereby filtering or pruning off irrelevant sub-trees. Making these choices while traversing the full depth of the tree is equivalent to selecting a path within a decision tree, building a set of dimensions and attributes (one pair for each node in the path) that best represents the usability problem and its causes.

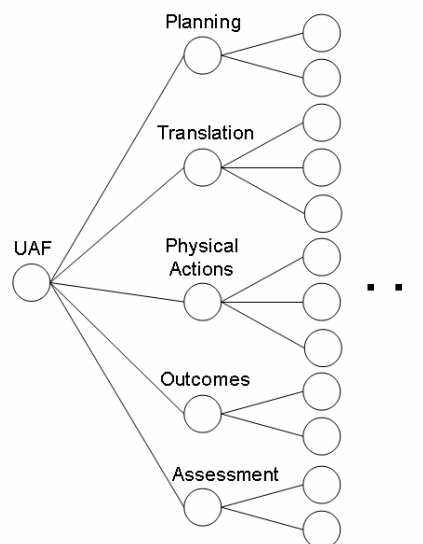


Figure 4: User Action Framework as a tree structure

Once a usability problem has been associated with a node, the path to that node contains all the information needed to identify the usability problem specifically. Precision is enhanced by the standardized usability vocabulary used. Reliability is enhanced because other usability problems that have the same attributes will be placed in the same node, and completeness is enhanced because the process leads the problem analyst to include all the relevant usability attributes.

2.1.2.2 Partial Diagnosis

Full diagnosis with the UAF can be time consuming, and it is not practical to try to diagnose usability problems during a session with a participant. Trying to perform full diagnosis by reviewing screen capture video after the session when the participant is gone, however, may also not be possible, especially if the necessary information for making a decision among multiple diagnoses is known only to the participant. It is therefore necessary to capture the right information about what a participant is doing or trying to do, which we refer to as *immediate intention*, during the usability data collection stage to enable complete and consistent diagnosis in the usability problem analysis stage. We propose modifying the usability evaluation sub-process to support a non-sequential, micro-iterative usability data collection and analysis process that we refer to as *micro-iteration*, which helps facilitators identify and capture the usability data needed by problem analysts to accurately and consistently diagnose problems.

2.1.2.3 Immediate Intention

Unlike medical doctors who have a structured diagnostic framework to help them determine what questions to ask and tests to run, problem analysts often cannot know which diagnostic questions need answering until beginning the analysis stage, after the participant is typically gone. Our exploratory studies in (described in section 5) suggested that these key early diagnostic questions involve very specific details about what the participant was doing or attempting and why at the time of experiencing a usability problem. We refer to these key details as the user's or participant's *immediate intention*, expressing them in terms of the type of user action involved (e.g., sensory, cognitive, physical) in the context of the location within the Interaction Cycle of the UAF (e.g., Planning, Translation, Assessment).

The UAF provides the necessary structure for determining which diagnostic questions apply and whether the appropriate data has been collected to completely specify immediate intention. Selecting a top-level node of the UAF completely specifies the kind of action that the participant was doing or attempting when he or she encountered an interaction design flaw. Understanding a participant's immediate intention therefore involves getting the data to distinguish among stages of the Interaction Cycle. Immediate intention allows designers to select an appropriate solution from a number of possible solutions. In some situations, one solution will fix usability problems with different immediate intentions. In other situations, however, usability problems with different immediate intentions have very different fixes. The following example illustrates this point.

A digital library website has a variety of tabs at the top of every page that serve as a navigation bar. A participant had trouble using the site to locate a specific journal because tabs associated with information-seeking tasks are mixed with those associated with other tasks. A possible solution to this usability problem is to reorder the tabs, so that tasks of a similar nature are adjacent to one another. This solution is sufficient if the participant had already planned for the task and was simply trying to determine which tab to select. In such a case, the participant has an immediate intention that maps to the Translation stage because she had already formulated a goal and developed a high-level task sequence to accomplish that goal. If the participant was not in

the Translation stage, reordering the tabs may not be a sufficient solution. For example, if the participant was not familiar with digital library sites or with the functionality of the particular site being tested, she may not have formed a high-level task sequence before she experienced the usability problem. The participant's intention may have been to understand the site and determine possible uses. In such a case, the participant was in the Planning stage of the Interaction Cycle when the usability problem occurred, and the tab ordering problem is a planning problem. An appropriate solution for a planning problem might require additional organization of the tabs, possibly into groups labeled by high-level task and workflow categories, accompanied by a link to an overview page with descriptions of functions.

As the example illustrates, the difference in immediate intention results in two different diagnoses with potentially two different solutions. Key details needed to distinguish between the Planning and Interaction stages of the Interaction Cycle are necessary to help the developers know which usability problem is the real one that occurred for the participant and, therefore, which solution is most appropriate.

2.1.2.4 Micro-Iteration

The exploratory studies (Section 5.1) helped us realize that it is necessary to capture key data during the usability evaluation to enable consistently correct diagnosis of usability problems. If important diagnosis questions cannot be answered with data captured while the participant is present during usability data collection, it is difficult or even impossible to answer them later. We concluded that it is necessary to perform some initial diagnosis during the evaluation while the facilitator is still able to communicate with the participant. This is a simple, but we believe, crucial conclusion, and it has reshaped our thinking about how to perform diagnosis. Having captured the necessary immediate intention information, the evaluator can perform full diagnosis after the participant is gone.

Including initial diagnosis may result in increased costs for the usability data collection stage because it requires keeping the participant for a longer period of time to establish and confirm immediate intention. The added cost, however, is the key to capturing the immediate intention information needed for usability problem analysis and usability information reporting. Without the correct information, later stages of the UE process cycle could potentially be less effective and more costly.

2.1.2.5 Diagnosis Wizard

We have developed two important concepts: immediate intention and micro-iteration. Immediate intention provides information about what the participant is doing when he or she experiences a Usability problem. Micro-iteration is a modification to the usability evaluation sub-process that gives facilitators the chance to ask questions of the participant during empirical evaluations or of themselves in analytical evaluations to get the data that are necessary to determine immediate intention. In this section, we introduce a tool that can be used during micro-iteration to help facilitators determine what to ask to specify immediate intention.

Evaluators need some kind of support in asking the right questions to elicit immediate intention information. The UAF has proved to be useful in structuring the process of capturing missing usability problem data, but the UAF is intended for use in the usability problem analysis stage and is probably too bulky and time-consuming for use by most evaluators for initial diagnosis as part of the usability data collection stage. As a result, we developed the Diagnosis Wizard, a lighter-weight tool that is limited to the top-levels of the UAF (the Interaction Cycle) and tailored

specifically for helping evaluators to quickly identify the immediate intention associated with a usability problem during micro-iteration.

The exploratory studies (section 5.1) helped us understand top-level diagnosis by allowing us to follow participants' thought processes while they tried to map usability problem descriptions to stages in the Interaction Cycle. The participants generally understood what was represented by the stages of the Interaction Cycle, but they had no process for comparing them. We noticed that when we coached participants at making this top-level diagnostic decision in the second exploratory study, it helped to break the multi-way decision down into a sequence of dependent two-way decisions, allowing the evaluators to focus on a single issue or question at a time. Encouraged by initial success with this approach, we codified it into a sequence of two-answer questions, each comparing one stage of the Interaction Cycle with the other stages, based on the distinguishing attributes of that stage. Each answer prunes the number of stages remaining. Through a process of elimination, the Wizard helps evaluators home in on the correct stage. If at any point the evaluator is unable to answer a question, she should interact with the participant to get the answer. The sequence is designed to first rule out the least likely stages of the Interaction Cycle and then continue to the most likely stages. Stages are ruled out in the following order: Outcome and System Functionality, Overall, Physical Actions, Assessment, Planning, and Translation.

Figure 5 depicts the ruling-out strategy. Each black node represents a decision point where the usability problem analyst chooses between a given stage in the Interaction Cycle and all the remaining stages. usability problem analysts start the Wizard by choosing between the Outcome and System Functionality stage and the rest of the Interaction Cycle.

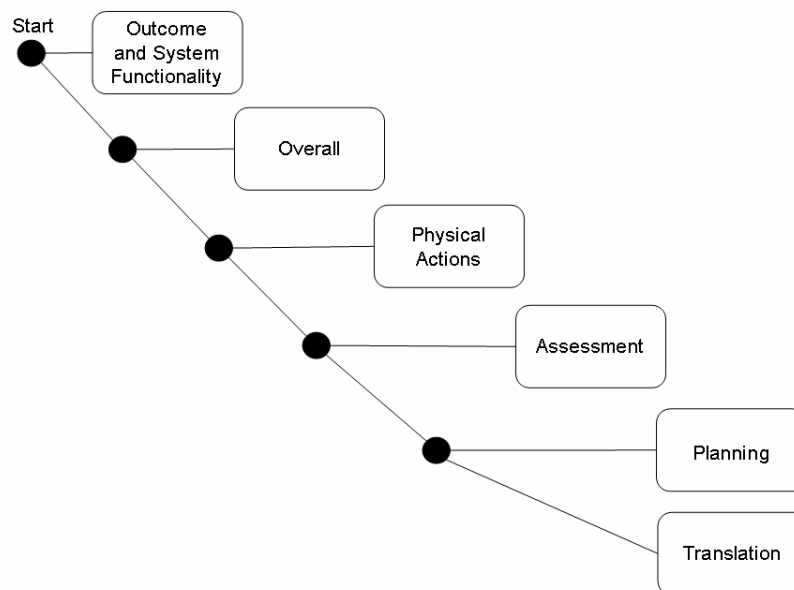


Figure 5. Wizard decision structure.

A *distinguisher* is a set of words that tersely captures the essential difference between the semantics of one UAF node and the semantics of the other nodes. For example, the text for the Physical Actions node in the Wizard is as follows: *“Is your problem about actually performing*

physical actions on interface objects or with devices? For example, does the user have problems finding or seeing an object to click or actually performing the clicking and dragging?” In this way the Wizard brings the right distinguisher to bear at the right time and the right place for the evaluator.

While the distinguishers needed are usually among the words in the UAF, most UAF nodes contain a description of the semantics of that node and not direct comparisons with other possible choices in sibling nodes. In contrast, the Wizard helps evaluators focus directly on the distinguishers by converting more verbose n-way UAF decision points into a series of more crisply stated binary questions based specifically on the differences between a given node and its siblings. At any one time, the facilitator can think about just one direct A vs. B face-off choice distinguished by a participant's immediate intention.

Section 5.3 documents formative evaluations that we performed with the Wizard. The results of the study suggest that the Wizard is useful in helping usability practitioners identify immediate intention.

2.1.3 Technical Specifications

The DCART component is written in C# and uses the Microsoft .NET Framework. It runs only under the Windows operating system. DCART can be configured to store data in local database files and in networked databases. DCART works with databases that support the ADO.NET OleDb provider. We use Microsoft Access for local database files and Microsoft SQL Server 2000 for networked databases.

2.1.3.1 Support for Context

Figure 6 shows support for levels of context and associated resources. The *levels view* in the top left hand corner shows the levels of context in a tree form. Nested contexts are represented as children of the parent context. The tree is expanded to show all the context levels; the letters to the left of the name of each node in the tree indicate the context level: organization (O), project (P), version (V), session (S), and task run (T). Clicking on a context level updates the *resource view* in the lower left hand corner and displays the level in the *workspace view* on the right side of the screen.

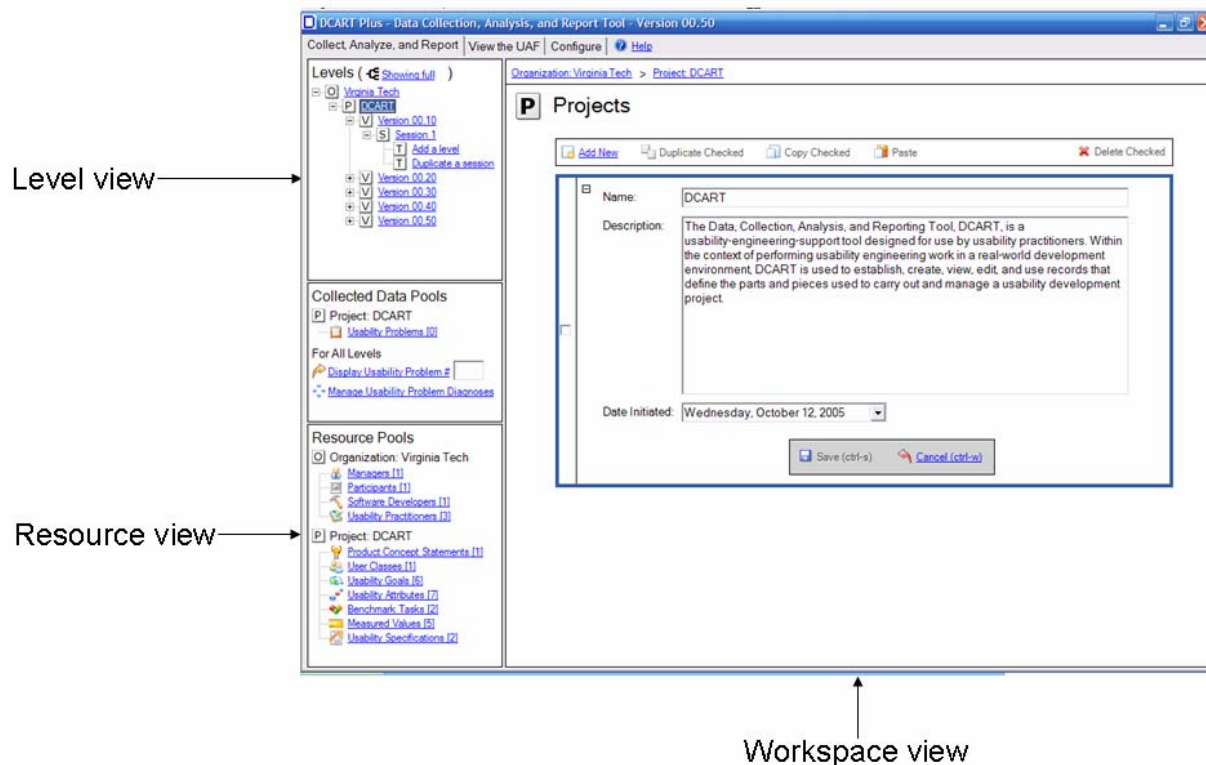


Figure 6. Support for levels of context and associated resources.

In Section 2.1.1.1, resources are described within the contexts in which they are used. In DCART resources are used in these same contexts, but they are pooled at higher levels of context to facilitate reuse. For example, although benchmark tasks are used at the task run level, they are pooled at the project level, so that they can be reused for multiple task runs of multiple sessions of multiple versions of the project.

The resource view has two lists of resources. The top list contains resources that are pooled in the organization level: managers, participants, software developers, and usability practitioners. The bottom list contains resources that are pooled in the project level: product concept statements, user classes, usability goals, usability attributes, benchmark tasks, measured values, and usability specifications. To the left of each resource is an icon that is used to represent the resource in other parts of the application, such as in usability problem records. As in the level view, selecting a resource will display it in the workspace view.

The resources that are pooled in the selected level of context are made available in the resource view. In Figure 6, the DCART project level is selected, so the resources pooled in the "Virginia Tech" organization and "DCART" project levels are available. The organization resource pool is available because projects are nested inside of organizations, and selecting a project implies selecting its parent organization. If the Virginia Tech organization were selected in the level view, then only the organization resource pool would be active, and the project resource pool would be grayed out.

Figure 7 shows the workspace view when a version level is selected. The workspace view consists of two parts. The first part at the top of the view shows the path of levels to the current

level. The second part is a control that we refer to as an expanding list. Records, individual rows in the expanding list, can be contracted and expanded. A contracted record, such as the record shown in the figure for “Version 00.20”, shows only the name of the level or resource that it contains. Clicking on the plus symbol or selecting the name text expands the record to show additional fields, as shown for “Version 00.10”. When a level is selected in the level view, it and all of its sibling levels are displayed in the expanding list. The selected level is initially expanded and the non-selected sibling levels are initially contracted. Resources exist in pools; selecting a pool of resources displays those resources in the expanding list. All the resources are initially contracted.

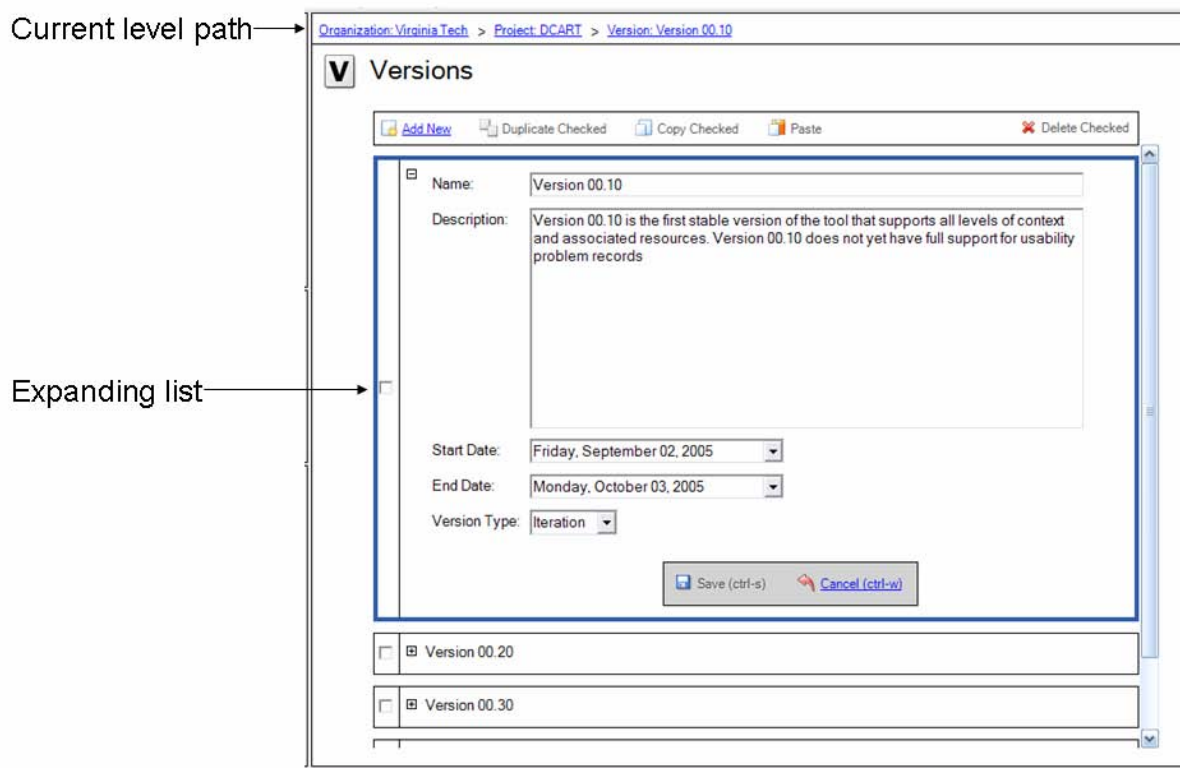


Figure 7. Workspace view.

All the levels and resources can be edited in place inside of an expanded record. Figure 8 shows an edited version of the data for “Version 00.10”. After a version has been edited, the background color changes, and the save link becomes active. If a record is saved, the background color turns back to white and the save link becomes grayed out. If the changes to a record are cancelled, the record is contracted. If changes have been made and another level or resource is selected, DCART prompts the user to save changes before loading the new expanding list.

Name: Version 00.10

Description: Version 00.10 is the first stable version of the tool that supports all levels of context and associated resources. Version 00.10 does not yet have full support for usability problem records. This version is essentially being used to test the usability of the expanding list control.

Start Date: Friday, September 02, 2005

End Date: Monday, October 03, 2005

Version Type: Iteration

[Save \(ctrl-s\)](#) [Cancel \(ctrl-w\)](#)

Figure 8. Edited expanding list record.

In addition to editing the fields of individual records, the expanding list control can be used to modify records. Figure 9 shows the record modifications option bar at the top of the expanding list control. The “Add New” option is always active; selecting this option will create a new record in the list and expand it. When individual records are selected using the selection checkbox, the appropriate options on the option bar become active. For example, the record for Version 00.10 has been selected, so the “Duplicate Checked”, “Copy Checked”, and “Delete Checked” options are active. Selecting any of these options will perform the requested action using the selected record as the target. The “Duplicate Checked” and “Delete Checked” options work on multiple selected records.

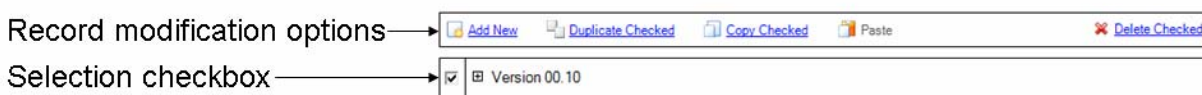
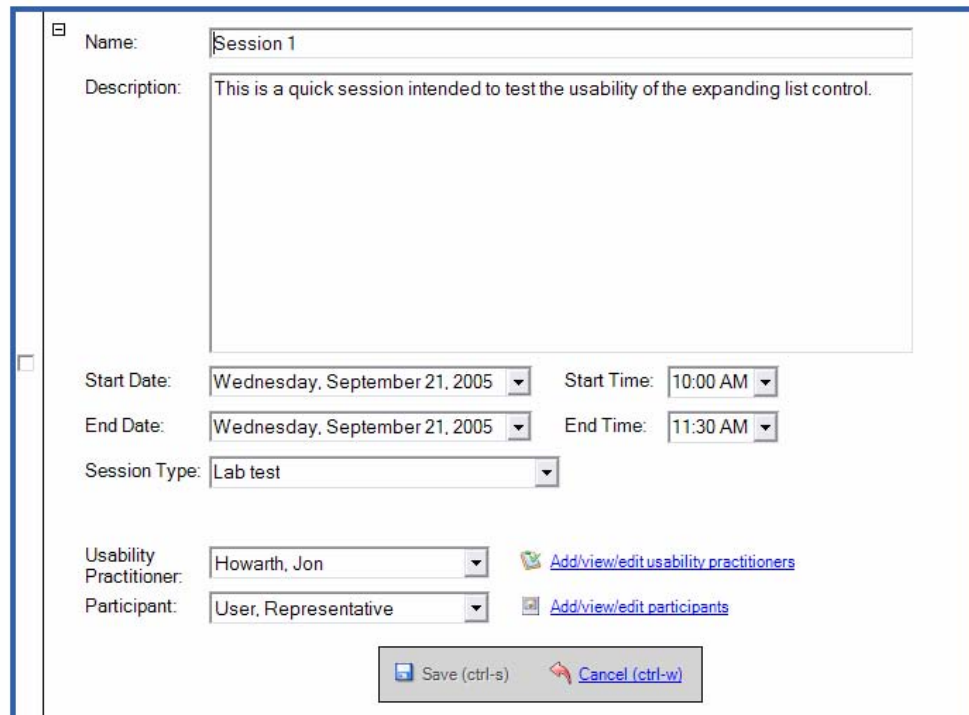


Figure 9. Modification options bar and selection checkbox.

2.1.3.2 Support for Format

Usability problems are identified and recorded during usability sessions. A session can consist of a usability practitioner observing a participant or performing an inspection or an expert walkthrough. Session levels exist inside of a given organization, project, and version. Figure 10 shows a session record for a session that was run with a participant to evaluate “Version 00.10”. The usability practitioner and participant shown in the record are selected from the associated resource pool at the organization level.



A screenshot of a web-based form for recording a session. The form is titled 'Session 1' and contains several fields for data entry. The 'Name' field is filled with 'Session 1'. The 'Description' field contains the text 'This is a quick session intended to test the usability of the expanding list control.' Below this, there are fields for 'Start Date' (Wednesday, September 21, 2005), 'Start Time' (10:00 AM), 'End Date' (Wednesday, September 21, 2005), and 'End Time' (11:30 AM). The 'Session Type' is set to 'Lab test'. There are also dropdown menus for 'Usability Practitioner' (Howarth, Jon) and 'Participant' (User, Representative). To the right of these dropdowns are links: 'Add/view/edit usability practitioners' and 'Add/view/edit participants'. At the bottom of the form are 'Save (ctrl-s)' and 'Cancel (ctrl-w)' buttons.

Name:	Session 1		
Description:	This is a quick session intended to test the usability of the expanding list control.		
Start Date:	Wednesday, September 21, 2005	Start Time:	10:00 AM
End Date:	Wednesday, September 21, 2005	End Time:	11:30 AM
Session Type:	Lab test		
Usability Practitioner:	Howarth, Jon	Add/view/edit usability practitioners	
Participant:	User, Representative	Add/view/edit participants	
<div>Save (ctrl-s) Cancel (ctrl-w)</div>			

Figure 10. Session record.

During each session, participants perform a given number of tasks. Figure 11 shows a task run record for one of the tasks for the session in Figure 10. The user class, benchmark task, and usability specification shown in the record are selected from the associated resource pool at the project level.

The screenshot shows a web-based form titled 'Task run record' under the 'Collect and Review' tab. The form is divided into several sections:

- Name:** A text input field containing 'Add a level'.
- Description:** A large text area containing the text: 'Participants are asked to add a new project. We are trying to identify how well participants understand the hierarchy of levels and how well they can manipulate them.'
- User Class:** A dropdown menu showing 'Usability Engineer'. To its right is a link: 'Add/view/edit user classes'.
- Benchmark Task:** A dropdown menu showing 'Add a new project'. To its right is a link: 'Add/view/edit benchmark tasks'.
- Usability Specifications:** A checkbox labeled 'Initial understanding of levels in the hierarchy' is checked. To its right is a link: 'Add/view/edit usability specifications'.
- Buttons:** At the bottom, there are two buttons: 'Save (ctrl-s)' and 'Cancel (ctrl-w)'.

Figure 11. Task run record.

Each task run consists of two steps: collecting usability problems in the form of usability problem records and reviewing the collected usability problem records to fill in necessary details. The evaluator does the first step of collecting usability problems while the participant is performing the task. When the participant is finished with the task, the evaluator reviews the usability problem records that she created and adds additional notes or observations that she did not have time to record during the running of the task.

Figure 12 shows the first step; each task run has an option under the “Collect and Review” tab that starts a form, which is used to create usability problem records during the task run. The form has four separate areas. The top left corner shows the context including the usability practitioner, participant, user class, benchmark task, and usability specifications. The evaluator can select any of these resources during the task run to see the resource’s full record in a separate window. Below the context area is an error counter that evaluators can use to tally errors committed by a participant; not all errors indicate usability problems. The time on task area below the error count area is a manual timer that evaluators can use to record the amount of time that a participant is actively involved in the task. The timer can be paused to account for interactions that are not part of the task, such as further explaining task instructions.

Figure 12. Usability problem collection form.

The final area on the right is the usability problem collection form. It is designed to allow evaluators to quickly create usability problem records as participants experience usability problems during the task run and contains the basic fields needed to capture the essence of a usability problem. During a task run, the evaluator uses the Ctrl-n hotkey combination or the “Save and add new” link to create a new usability problem record for each usability problem encountered by the participant. The evaluator briefly documents each usability problem by giving it a name and a brief description.

After the participant has performed the task, DCART displays a brief summary of the task under the “Collect and Review” tab of the trial record. The evaluator then selects an option that opens a form used for the second step of reviewing the collected usability problem records (Figure 13). The form is similar to the form used to collect usability problems except that it provides a way for evaluators to iterate through the collection of usability problem records. The evaluator uses this form to review usability problem descriptions and fill in details.

Review Usability Problems (while the participant is present)

Project Context

- Usability Practitioner: [Howarth, Jon](#)
- Participant: [User Representative](#)
- User Class: [Usability Engineer](#)
- Benchmark Task: [Add a new project](#)
- Usability Specifications: [Initial understanding of levels in the hierarchy](#)

Usability Problem Review Form

[Add new](#) [Delete](#)

Problem ID: 1

Problem Start Time: 1:05:28 PM

Problem End Time: 1:09:02 PM

Problem Name:

Problem Description:

Immediate Intention:

- Stage of Interaction Cycle: [Use Wizard](#)
- Type of Action:

Error Count: 2

Time on Task: 00:02:38

Navigation: << First < Previous Problem 1 of 4 > Next >> Last

Figure 13. Usability problem review form.

The usability problem records created during the task run are made accessible through the data view shown in Figure 14. When a task run is selected in the level view and the “Usability Problems” option is selected in the data view, all the usability problem records associated with the task run are displayed in the workspace view. The usability problem records are shown in an expanding list.

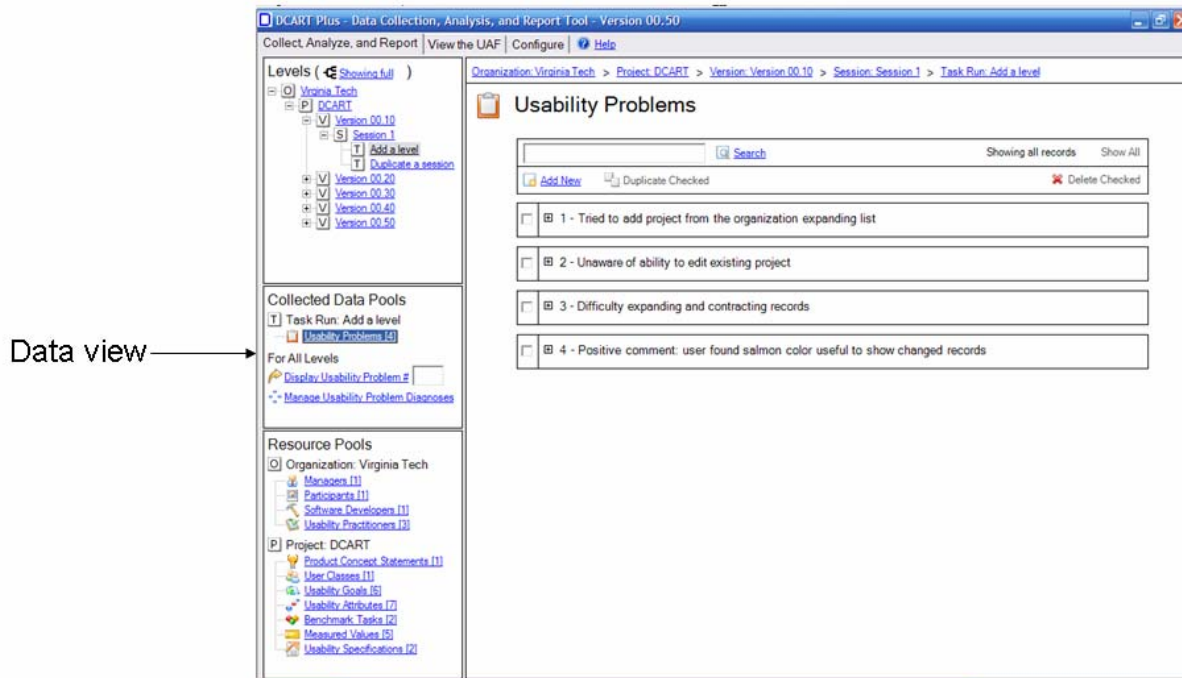


Figure 14. Data view.

Figure 15 shows an expanded usability problem record. The “Project Context” section at the top contains all the information that appeared in the usability problem collection and record review forms as well as information about the time at which the usability problem was encountered in the task run. The “Usability Problem” section below the context section contains a number of fields that can be used to describe and specify the usability problem. Each usability problem record is assigned a problem id. The information entered in the usability problem collection and record review forms is included in the name and description fields of the usability problem record. The other fields in the usability problem report are filled out after the participant has left and are used to document the user interface object or objects associated with the usability problem, designer knowledge about how the design should work, immediate intention, UAF diagnosis, diagnosis status, solution suggestions, cost, importance, and any other useful comments.

Project Context

Virginia Tech > DCART > Version 00.10 > Session 1 > Add a level

Usability Practitioner: [Howardh_jon](#)

Participant: [User_Representative](#)

User Class: [Usability Engineer](#)

Benchmark Task: [Add a new project](#)

Usability Specifications:
[Initial understanding of levels in the hierarchy](#)

Task Run Start Time: 1:05:28 PM

Task Run End Time: 1:11:01 PM

Problem Start Time: 1:05:28 PM

Problem End Time: 1:09:02 PM

Usability Problem

Problem Id: 1

Name:

Description (including essence, type, causes, and usability principles):

The participant attempted to add a project by selecting the "Add New" modification option from the organization level.

Image (screenshot or illustration of the problem - accepts bmp, gif, jpg, png, and tiff formats):

[Add Image](#) [View Image](#) [Save Image as File](#) [Delete Image](#)

User interface object (what screen, button, menu, dialogue box, etc. is the main focus of the problem):

Designer knowledge (interpreting observations by knowledge of how design is supposed to work):

Immediate Intention (what the user was doing or attempting at the time of the problem):

- Stage of Interaction Cycle (top level of the UAF):

Planning

[Use Wizard](#)

- Type of action:

Cognitive

Diagnosis: [Diagnose with the User Action Framework \(UAF\)](#)

Diagnosis database: None

Status (stage of completion of the usability problem report):

Solution (including usability engineering and software engineering perspectives):

Importance (M = must fix or 1 = low importance to 5 = high importance):

Comments (including recommendations for further evaluation, additional rationale):

Save (ctrl-s)

Cancel (ctrl-w)

Figure 15. usability problem record.

2.1.3.3 Support for Diagnosis

In this section, we first discuss DCART's support for full diagnosis with the UAF, then we describe how DCART supports micro-iteration to help evaluators capture immediate intention for partial diagnosis.

2.1.3.3.1 Support for Full Diagnosis

The UAF can be viewed and navigated within DCART. Selecting the “Diagnose with the User Action Framework (UAF)” option inside of usability problem records (Figure 15), opens the UAF diagnosis form in a new window (Figure 16).

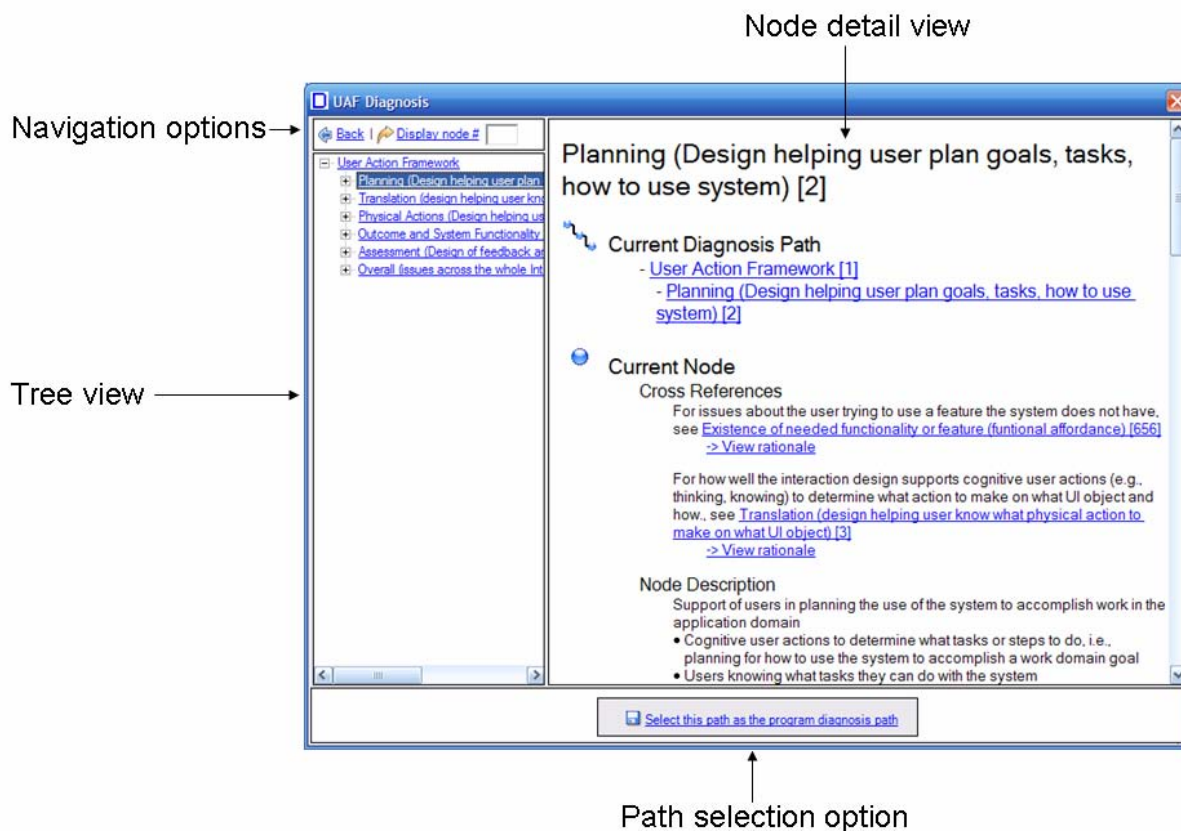



Figure 16. UAF Diagnosis form.

Figure 16 shows the four major areas of the UAF diagnosis form. The top left corner contains navigation options to allow an evaluator to go back or jump directly to a node with a given node number. The tree view on the left-hand side allows practitioners who are familiar with the UAF to quickly traverse it. Practitioners that are not familiar with the UAF can traverse the tree using the node detail view. The tree view uses minus signs for expanded nodes, plus signs for expanding nodes with children, and empty boxes for terminal nodes. Selecting the link for a node in the navigation tree will display the content of that node in the node detail view. Selecting the box to the left of the hyperlink will perform the appropriate action on the navigation tree, such as expanding a node with a plus, without refreshing the node detail view. The path selection option


below the tree view allows evaluators to select the current path as the diagnosis path for their usability problem. When a path has been selected, the window closes and the path is inserted into the usability problem record.

Figure 17 shows the node detail view. The first item is the name of the node, with the node's unique id displayed in brackets at the end of the node's name. Below the name is a representation of the current diagnosis path in a horizontal tree similar to the tree view. The next item is the current node, which contains cross references, a node description, and examples. The final item is a listing of children of the current node. The Planning node shown in Figure 17 actually has eight children, but we limit the screenshot to two children.

Planning (Design helping user plan goals, tasks, how to use system) [2]


Current Diagnosis Path

- [User Action Framework \[1\]](#)
- [Planning \(Design helping user plan goals, tasks, how to use system\) \[2\]](#)


Current Node

Cross References

For issues about the user trying to use a feature the system does not have, see [Existence of needed functionality or feature \(functional affordance\) \[656\]](#)
-> [View rationale](#)

For how well the interaction design supports cognitive user actions (e.g., thinking, knowing) to determine what action to make on what UI object and how, see [Translation \(design helping user know what physical action to make on what UI object\) \[3\]](#)
-> [View rationale](#)


Node Description

Support of users in planning the use of the system to accomplish work in the application domain

- Cognitive user actions to determine what tasks or steps to do, i.e., planning for how to use the system to accomplish a work domain goal
- Users knowing what tasks they can do with the system
- How well the system supports learning about the system for planning

Examples

- If users cannot determine how to organize several related tasks in the work domain, because the system does not help them understand exactly how the system can help do these kinds of tasks, the system design could be thought to be the cause of these users' planning problems.


Children of Current Node

- 1. [User high-level understanding of system \[9\]](#)**

Node Description

Support of user ability to achieve high-level understanding of the system, especially the interaction design

 - Issues about users' high-level understanding of the whole system, not just how to use one feature
 - Overall user understanding of design concept
 - Awareness of how a user can use specific system features
 - User overall expectations
 - User-centered representation of planning-oriented user tasks

Examples

None
- 2. [Goal decomposition \[10\]](#)**

Node Description

Support of user task decomposition, logically breaking long, complex tasks into smaller, simpler pieces

 - System should accommodate human memory limitations in dealing with complex goals and tasks.
 - System should avoid stacking and interruption.
 - System should provide task and subtask closure as soon as

Figure 17. Node detail view.

In the current node item, cross references appear first to immediately redirect evaluators who have incorrectly arrived at the node. Each cross reference contains two pieces of information: the high-level cross reference description of the target node and the rationale. The high-level cross reference description is pulled from the target node for consistency; because each node is cross referenced with the same text, practitioners can quickly identify key nodes and what distinguishes one from another. The rationale is specific to the current node's relationship to the cross referenced node and tells the practitioner why the cross referenced node may better describe the usability problem. In this view, the rationale is hidden and must be displayed with the "View rationale" option to limit the amount of information that a practitioner must initially process.

The node description and examples are displayed under the cross references. The node description consists of a brief overview that describes the node at a high level and bullet items that contain more detailed descriptions. One of the bullet items may be designated as a look-ahead description bullet that is displayed when the current node is displayed in the listing of children for its parent node. The look-ahead description bullet helps to guide practitioners down a particular path to a node. The examples are usability problems that would be classified in the node. Like description bullets, an example may be classified as a look-ahead example.

The children of the current node appear after the current node item. Each child is displayed with the high-level description and description bullets, including any look-ahead description bullets from its children. Examples are not displayed with the children to minimize display space.

2.1.3.3.2 Support for Partial Diagnosis

As discussed in Section 2.1.2, diagnosis with the UAF may be time consuming. DCART provides support for micro-iteration and capturing immediate intention through a two step process for identifying and recording usability problems that is described in Section 2.1.3.2. During the first step, the evaluator observes the participant and creates usability problem records using the usability problem collection form. The form has fields for immediate intention information (Figure 12). If the evaluator is unsure of the immediate intention, she leaves the field blank. During the second step, the evaluator reviews the usability problem records while the participant is still available and asks questions of the participant to elicit necessary information to determine immediate intention. The usability problem review form (Figure 13) has a "Use Wizard" option that opens the Diagnosis Wizard in a new window (Figure 18).

The Wizard window has a blue title bar with the text 'Wizard' and a close button. Below the title bar is a star icon and the word 'Wizard'. The main text area contains instructions: 'Please select the box that best describes your problem. If your selection maps directly to an immediate intention, it will be set as the immediate intention for your usability problem. If your selection helps to eliminate a stage of the Interaction Cycle but still does not map directly to a particular stage, you will be presented with another selection of a similar form.' Below this is 'Selection step 1 of 5'. There are two side-by-side boxes for selection. The left box contains the text: 'Is your problem in the non-user interface software (e.g., a bug in the back end computation)?' and 'For example, does the system automate too much and take control away from the user?'. The right box contains the text: 'Does your problem concern the user's interaction with the user interface?' and 'For example, is your problem related to user planning, determining actions, making actions, or understanding feedback?'. At the bottom, there are two buttons: 'Select Outcome and System Functionality' and 'Continue'. A link 'Go back to the previous set of questions' is centered at the bottom.

Figure 18. The Diagnosis Wizard.

As described in Section 2.1.2.2, the Wizard helps evaluators determine immediate intention by breaking down the decision process into a sequence of two-answer questions, each comparing one stage of the Interaction Cycle with the other stages, based on the distinguishing attributes of that stage. The Wizard in Figure 18 has two boxes. The left box describes a stage of the interaction cycle and the right box describes the remaining stages that have not already been ruled out. If the evaluator selects the option below the left box, then the Wizard window will close and the corresponding stage of the Interaction Cycle will be selected in the usability problem record review form. If the evaluator chooses the “Continue” option below the right box, the stage of the Interaction Cycle described by the left box is ruled out and the process is repeated with the next stage of the Interaction Cycle. If the evaluator does not know enough about the usability problem to choose one of the options, she should ask the participant questions to elicit necessary information.

2.1.3.4 Usability Problem Inspection

The Software Therapist system is also designed to support usability inspections. When practitioners perform an inspection, they perform essentially the same steps that they would perform for lab-based testing. One difference is that they do not specify a separate participant at the session level. A second difference is that they can use the UAF to help search for problems instead of using it to diagnose problems. This can be accomplished by browsing the UAF or with the assistance of the LSA-based search features provided in the ST Browser.

2.2 The Software Therapist Browser

2.2.1 Purpose & Design of the ST Browser

The purpose of the Software Therapist (ST) Browser is to support usability engineering practitioners as they engage in the tasks of analyzing and finding solutions to usability problems. This functionality complements and partially overlaps that of the DCART component of the Software Therapist system. For example, after setting up a usability data management database in DCART with organizations, projects, usability specifications, benchmark tasks, etc., and having conducted usability trials to collect data, practitioners may continue with usability problem diagnosis within DCART, for example by using the Diagnosis Wizard (Section 2.1.2.5). Alternatively, users may pursue problem diagnosis, further analysis, and possible solutions within the ST Browser, which provides support for LSA-based semantic search, as well as browsing, of the UAF, the problem report database, and any number of associated electronic library resources that may be pertinent to the usability issues under investigation. In this section, we will describe the ST Browser in detail and give some examples of its use.

The ST Browser is provided to its users (expected to be usability engineers/practitioners, software developers, and students) as a web-based service. The primary user interface is a Java applet (see <http://java.sun.com/applets/>), which is accessed from a web page, called the “launch page”. This page is the main page displayed in Figure 19. The launch page contains code that checks whether the current user’s browser meets the requirements for running the main Software Therapist applet.⁵ The launch page includes links to web sites from which the Java JRE may be downloaded in case the user’s browser does not include the required Java JRE. It also includes links for some additional help pages. If the user’s browser meets the requirements, the launcher applet indicates this and provides a button that can be clicked to launch the ST Browser applet. When the user does this, a launcher window appears to indicate that the ST Browser is loading, also shown in Figure 19. Access to the launch page and to resources within the browser may be restricted in the usual ways provided by modern web servers, such as by usernames and passwords (Figure 20).

⁵ Most modern browsers include a Java JVM that meets these requirements (JRE version 1.4 or later), and newer versions can be downloaded at no cost over the internet.

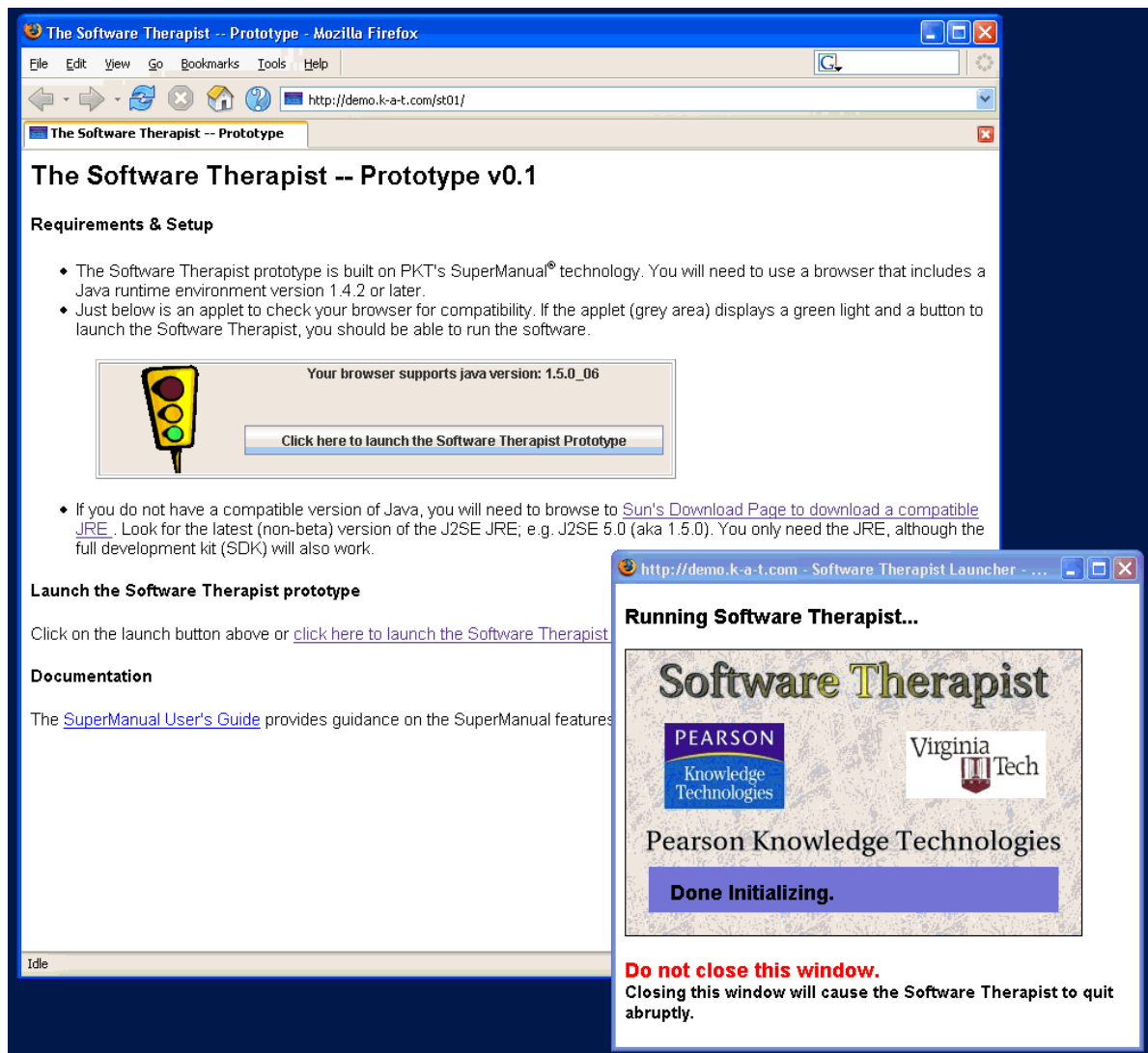


Figure 19. The Software Therapist (ST) Browser is launched from a web page.



Figure 20. A user management system can control access to different resources, such as integrated electronic copies of articles and textbooks.

After it has loaded, the ST Browser window appears, as shown in Figure 21. At this point, the browser is mostly empty, showing only the resources that are available for use from the server. The top left panel shows a folder icon labeled as “Library”, which indicates that the browser is displaying the contents of the available library. The available resources are listed in the panel on the right. In Figure 21, the items available are: the User Action Framework, the Problem Reports available in the database, and two HCI-related books in electronic format. The ST Browser also includes a user help file that explains the basic use of the interface, as shown in Figure 22.

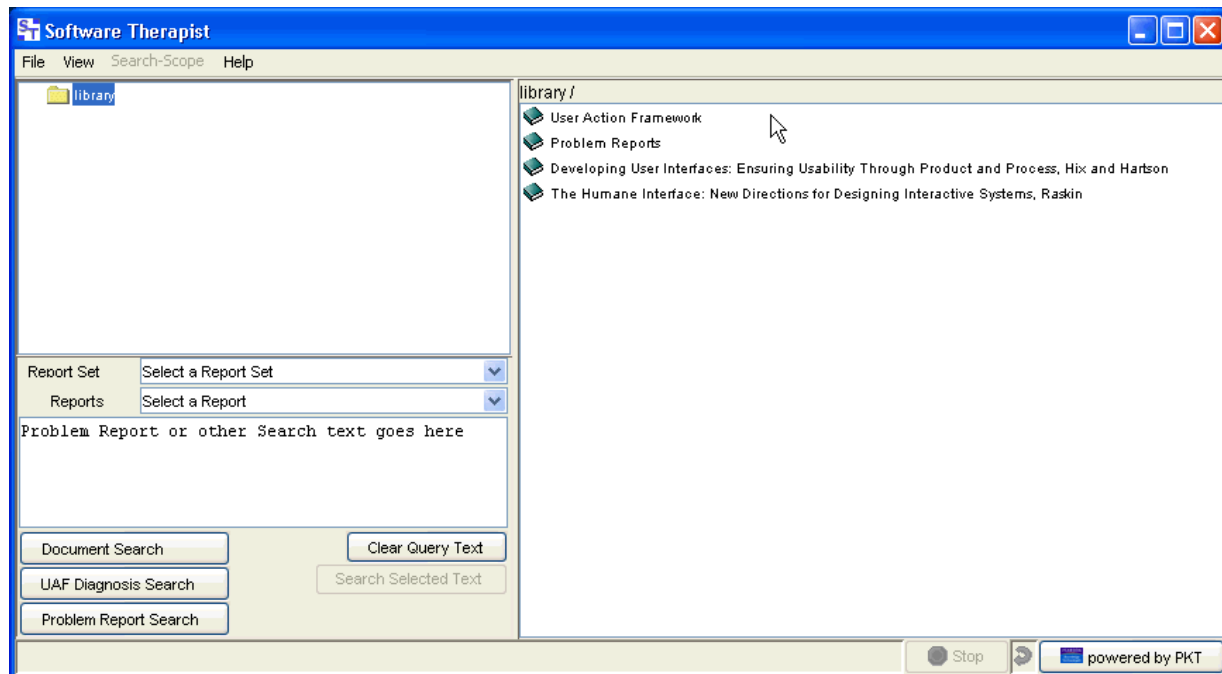


Figure 21. Software Therapist (ST) Browser, showing the Library View.

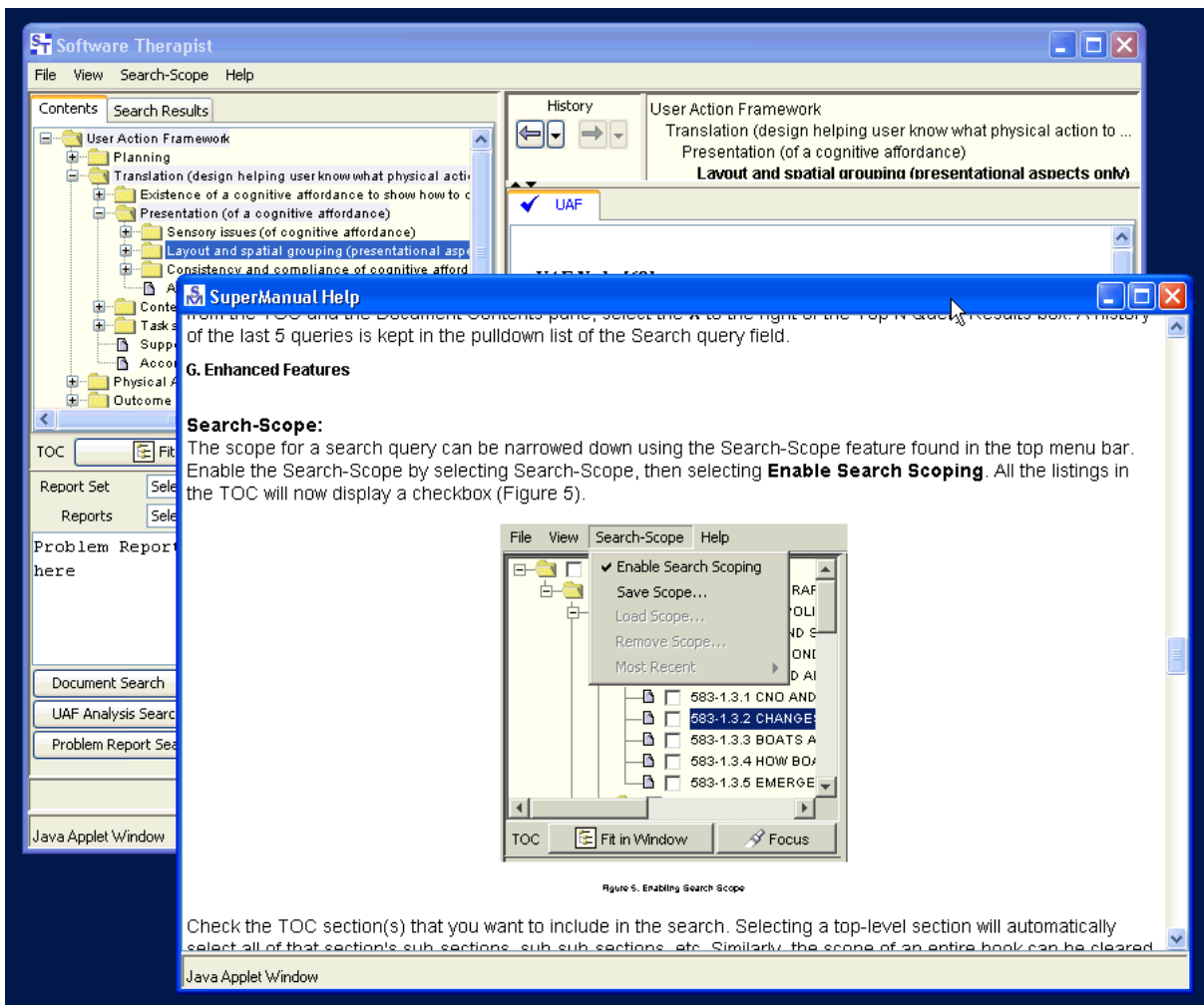


Figure 22. Help pages provide a guide to the ST Browser's features and how to use them effectively.

2.2.2 Using the Software Therapist Browser with the UAF

From the Library view, the user may open one or more of the available resources. Each will be displayed in its own tab, so users can quickly switch back and forth among them. Most of the business conducted in the ST Browser will involve the UAF, so we'll open that first by double clicking on the title "User Action Framework", which is highlighted in Figure 21. This loads the UAF into the browser and displays it as shown above in Figure 2 (and below in Figure 24 and following).

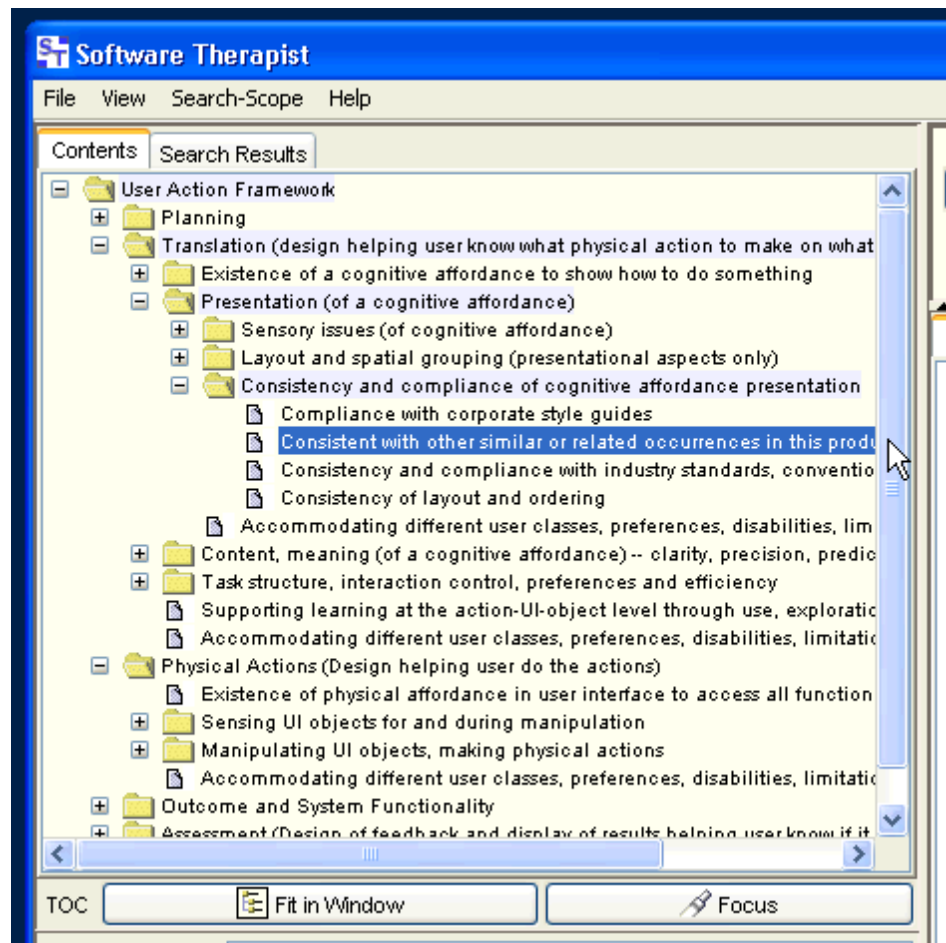


Figure 23. Detail of the TOC view.

At the top left of the window is the Table of Contents (TOC) panel, shown in detail in Figure 23. This displays the contents of the UAF (or other document) as an expandable tree structure. The full content of the currently selected UAF node is displayed as a scrollable page in the Main Display panel on the right, as in Figure 24, where the user has browsed to the UAF node titled “Sensory issues (of cognitive affordance)”. In the TOC, this node is highlighted, and each ancestor node is displayed with light highlighting. This indicates the *path* to the current UAF node. This path is also shown in the Path Summary panel at the top right. Each UAF node includes a title, a description, and (in most cases), one or more examples and cross references to related nodes that the user will likely want to consider when doing a problem diagnosis. Each of these is visible in the example in Figure 24.

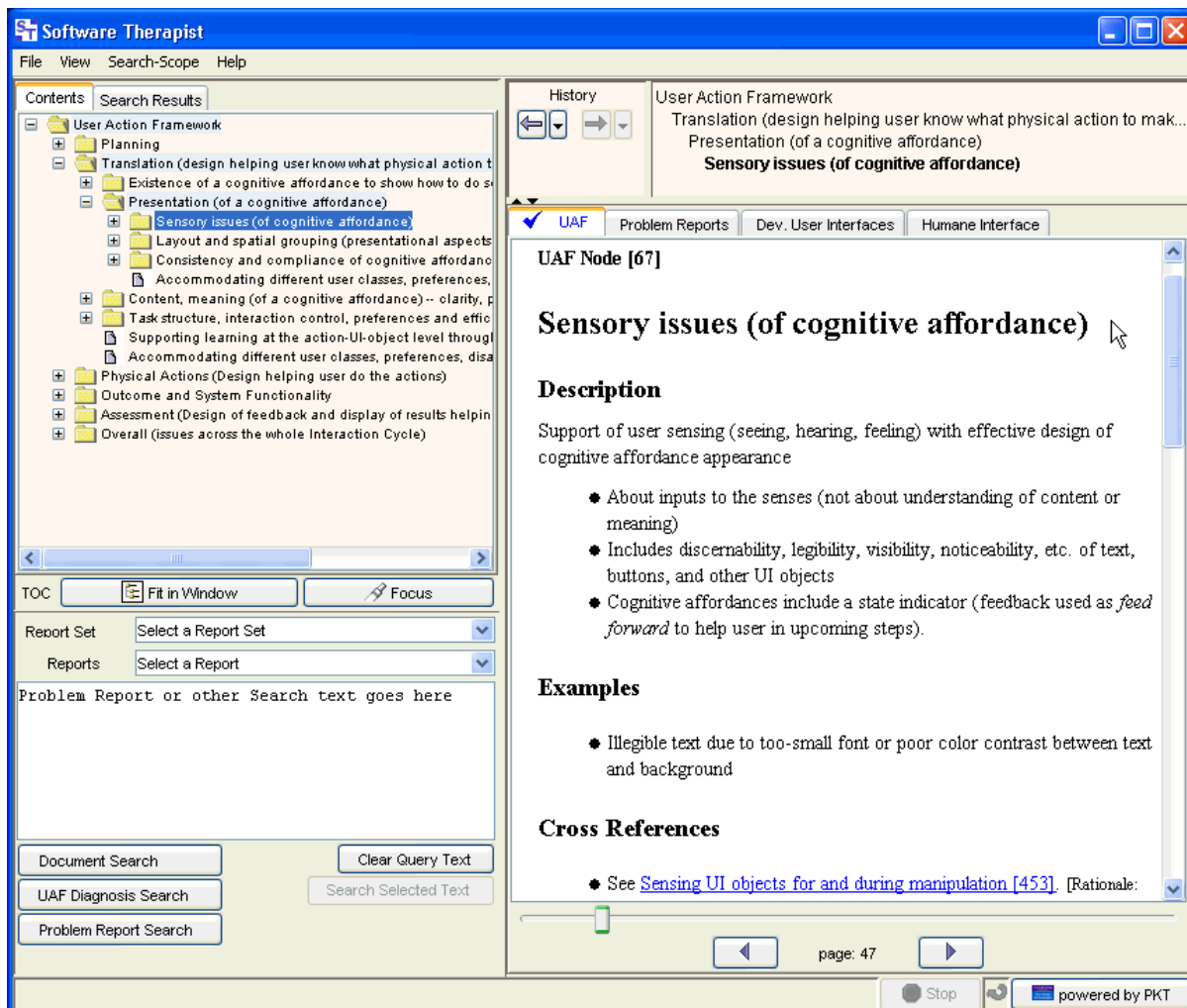


Figure 24. ST Browser showing the selected node highlighted in the TOC, the path to the current node shown in the Path Summary, and the node's content (title, description, examples, and cross-references) shown in the Main Display area.

The History buttons (top middle) allow the user to move forward and backward through the history of displayed nodes, either one step at a time or jumping to an arbitrary point in a pop-up list of history items, as in Figure 25. In addition, users may move one page forward or backward by clicking on one of the Page buttons or use the Page Slider to jump forwards or backwards in the UAF. These controls (at the bottom right) are of greater utility when browsing the online related literature, as the UAF (where each node is a page) is not generally intended to be read sequentially. (See below, e.g., Figure 42 to Figure 47.)

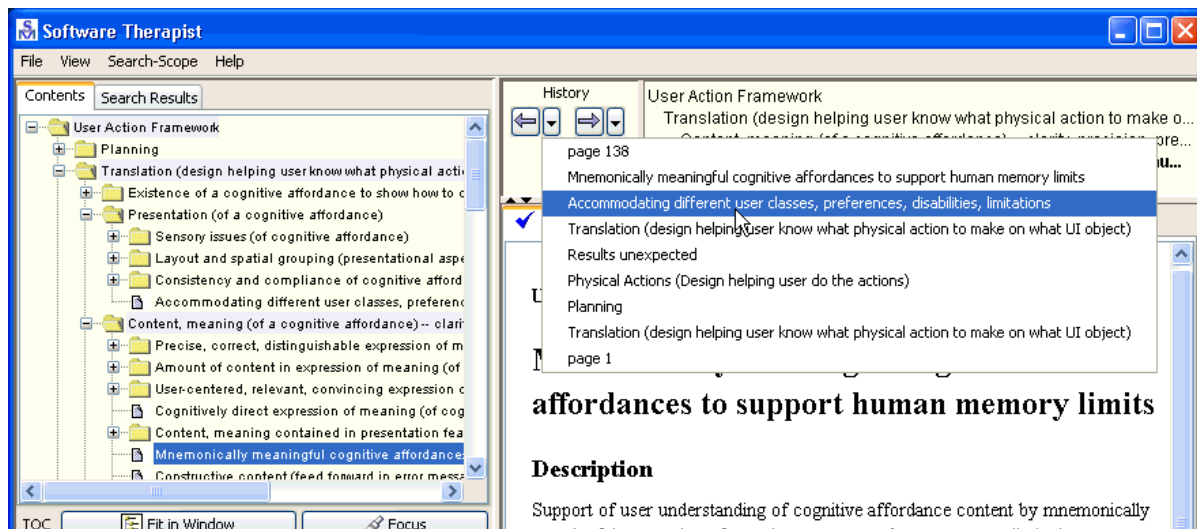


Figure 25. Navigation by browsing history.

All items displayed in the TOC and the Path Summary panels are active links, enabling the user to navigate easily along the current UAF path or (in the TOC) to neighboring or other nodes. This is particularly useful when doing problem diagnosis, as it allows the practitioner to move easily to any point in the diagnosis path to verify the decision made (UAF branch taken) at that point in the diagnosis. An example of this is shown in Figure 26, where the user has clicked in the Path Summary panel on an ancestor node titled “Presentation (of a cognitive affordance)”.

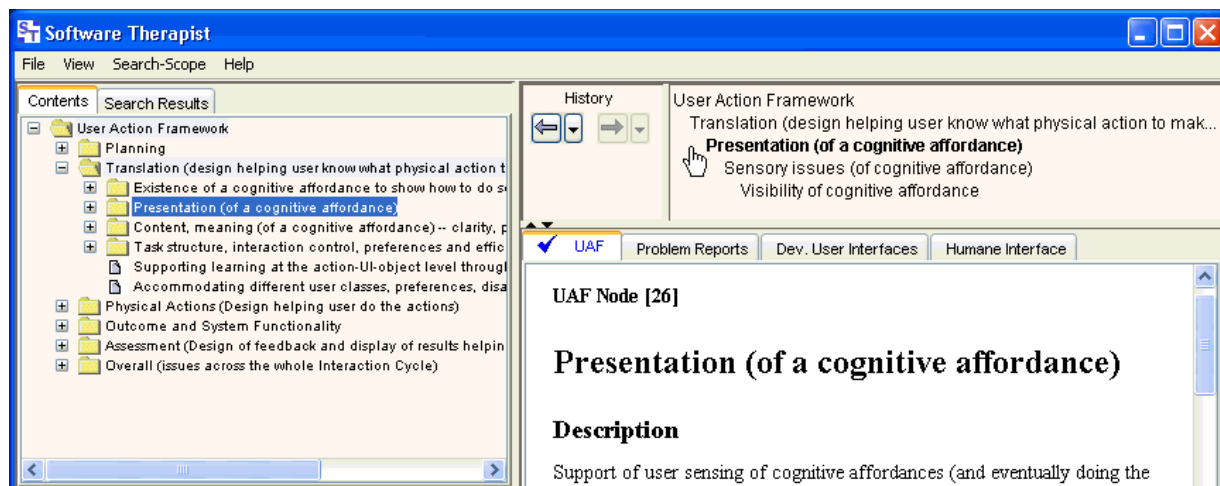


Figure 26. Navigation using the Path Summary panel.

2.2.3 Usability Problem Diagnosis in the ST Browser

The ST Browser can be used to as described above to navigate and browse the UAF for educational purposes or to analyze and possibly revised or extend the UAF. In addition, the UAF may be browsed for the purpose of diagnosing particular usability problems with the UAF. To do

this, the usability practitioner needs to examine a usability problem report and determine which path through the UAF constitutes the best diagnosis for that problem. To conduct this diagnosis manually, the practitioner begins at the root node of the UAF and, examining each of its child nodes (i.e. Planning, Translation, Physical Actions, Outcome, Assessment, and Overall), determines which branch should be followed at this decision point.⁶ Although a full problem diagnosis can be conducted by browsing one's way through the UAF, the ST Browser also provides several LSA-based search mechanisms that can help the user find an appropriate diagnosis. The ST Browser can also continue a partially diagnosed usability problem, for example when initial diagnosis has been made during a usability trial (or later) using DCART and the Diagnosis Wizard (section 2.1.2.5). How to narrow the scope of an LSA-supported search (e.g., to continue a partial diagnosis) is described below in section 2.2.3.

2.2.3.1 LSA and Semantic Spaces for Usability Analysis

The ST Browser's most important contribution to the process of usability analysis is its implementation of several varieties of search functionality based on Latent Semantic Analysis. The basic features and techniques used in LSA are described in Appendix A (section 8).

An LSA-based search consists fundamentally of a comparison of latent or "deep" semantic similarity among documents. A "document", for this purpose, is any natural language text, which can in principle be of any length, but which in practice is typically arranged to be of approximately paragraph length.⁷ The similarity comparison is made on the basis of latent/deep semantics, rather than on the basis of particular words ("keywords"). That is, two texts may be semantically similar without overlap in keywords (a synonymy effect), for example. The comparison can be made on this deeper semantic basis, because texts are compared using a *semantic space*, an artifact constructed from statistical properties of a large number of documents using a certain type of machine learning algorithm. The result of this training process is a large, high-dimensional vector space that represents—in an optimized and relatively compact, useful form—the co-occurrence statistics of all the words that appear in the documents used to derive the space. Each word (term) and each document (paragraph or similar passage) is represented as a vector in this space. In practice, LSA semantic spaces are typically constructed using a set of documents selected to be representative of the general topic of interest. If this set is representative of the domain of interest, the space can be used effectively to compare the semantic similarity of arbitrary texts in this domain—new documents (rather importantly), as well as those used in the construction of the semantic space.

For the application of LSA to usability analysis, we used a document base consisting of academic articles, book chapters, and several other texts from the domains of human factors, human-computer interaction (HCI), and usability engineering that were collected for this project. Also included were our collection of problem reports and the content of the UAF itself.

After collecting the documents to be used to construct the space, several pre-processing steps are applied to transform each document into an appropriate standard representation for construction of a semantic space.⁸ As mentioned above, when constructing a semantic space and using it to

⁶ This same diagnosis process can be conducted in the DCART component by browsing alone or with the guidance of the Diagnosis Wizard, as described above in section 2.1.2

⁷ See Landauer et al. (2004), Landauer et al. (1998).

⁸ For the purpose of building a semantic space, documents need to be transformed into a simple, raw text format. Sometimes this is relatively easy; other times quite hard. For example, many of the documents we used were originally available in Portable Document Format (PDF). (See <http://www.adobe.com/products/acrobat/adobe.pdf.html>.) In some cases, text can be extracted easily and

compare documents, one must decide on a definition for a single “document”. For our purposes here, we used a couple different specifications of what would constitute a document. For articles, books, problem reports, and other “normal” texts of this sort, we used paragraphs as naturally defined in the text itself as the basis for dividing the text in a set of passages (“documents” for LSA purposes). For the UAF, each node was considered a single document, as was the full text of each problem report in the problem report database.

After collecting the documents and pre-processing them, the semantic space can be built. This process involves building a term-by-document matrix that represents the occurrences of each term (word) in each document (paragraph, UAF node, etc.). A dimensionality reducing operation is then applied using Singular Value Decomposition (SVD).⁹ See Appendix A (section 8) and associated references for further details.

Using our usability document base, we created several versions of a usability semantic space during the project. The final space contained 24,100 (approximately paragraph-size) documents from about 470 original texts (the articles, book chapters, problem reports, etc. that constituted the usability document base). After the SVD operation, the resulting space has 315 dimensions. (Approximately 300 dimensions has proved to be an effective size for many LSA applications; see Landauer et al. (1998), Landauer et al. (2004).)

After a semantic space has been built, it can be used to compare the semantic similarity of terms and documents. For example, a particular word or phrase can be compared to each UAF node to see which nodes are most related to the concept that word represents, or, conversely, to find which words are most representative of particular UAF nodes, which may be useful when editing and extending the UAF. (See section 3.2.)

Or, for conducting a usability problem diagnosis, the text of a problem report can be compared to text from the UAF or to available literature in order to find articles or other treatments related to that problem report. To support usability analysis and related activities (such as editing the UAF), the ST Browser provides three varieties of LSA-based search. Each of these uses the usability semantic space to search for different types of documents related to an arbitrary text (such as a problem report).

The first type of search performs a standard LSA search for related passages (e.g. paragraphs) in whichever document is currently displayed in the ST Browser. If this is a book about usability engineering, this search will return a list of passages, ordered from most to least related to the query text, as determined by LSA. If the displayed document is the UAF, this search will find the most semantically similar nodes, as nodes are treated as individual passages in the UAF. This type of search we will refer to as “Document Search”.

The second type of search is called a “UAF Diagnosis Search”. In this case, LSA searches for paths (sequences of nodes in a parent-child relationship) through the UAF that are semantically similar to the query text (typically a problem report). That is, the problem report or other text entered by the user into the query box in the ST Browser is compared to each possible UAF diagnosis, where a diagnosis is defined as the relevant text of each node in a path through the UAF. The results are a ranked list of these paths from most to least similar, according to LSA on the basis of the usability semantic space. Examples are given below.

accurately from the PDF file; in other cases this is difficult and/or results in poor quality text, i.e., with many errors. Similarly, html files have markup stripped, leaving just the text. Finally, this text is processed to standardize the handling of upper- vs. lower-case, punctuation, etc.

⁹ See, for example, <http://mathworld.wolfram.com/SingularValueDecomposition.html>, http://en.wikipedia.org/wiki/Singular_value_decomposition.

The third type of search made available to the user of the ST Browser is a “Problem Report Search”. This is the same as a simple search, but with the set of problem reports in the usability database as the object of the search.

For each type of search, the scope of the search may be limited to designated portions of the tree structure represented by the Table of Contents for the currently displayed resource.

The use of these different types of LSA-based search in the ST Browser will be illustrated below.

2.2.3.2 Using LSA for Problem Diagnosis

To begin a diagnosis using the ST Browser, the user first selects a usability problem report. At the bottom left of the ST Browser is the *search panel*, which includes (1) controls for selecting a *problem report set* from the problem report database and selecting a particular *problem report* from that set, (2) a *search query box*, in which the problem report (or any other text) is displayed and can be edited, and (3) search control buttons for initiating one of several types of LSA-based searches on the contents of the query box or clearing the contents of the query box (labeled “Search”, “UAF Diagnosis”, “Clear”). Figure 27 shows the user selecting a problem report set to work with, one titled “NCCTRL”.

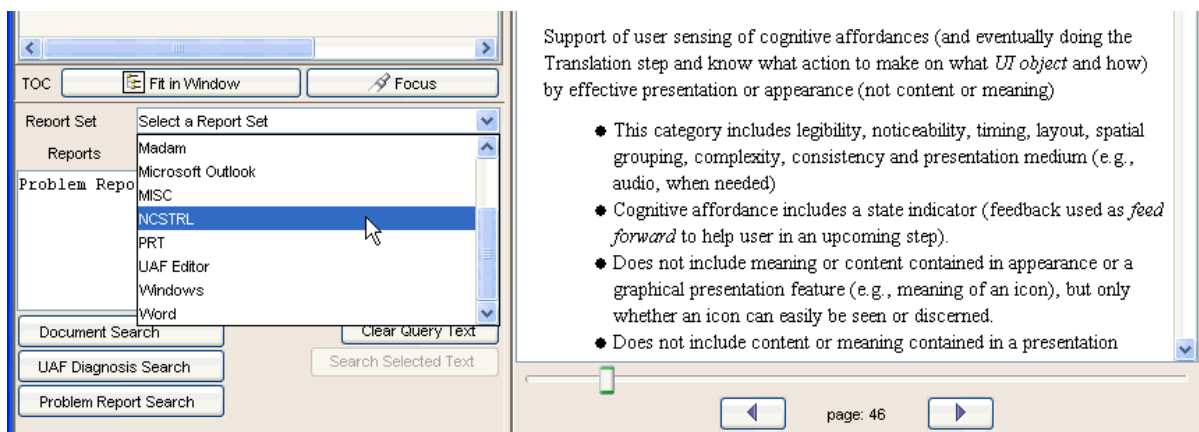


Figure 27. Selecting a Problem Report Set.

After selecting a problem set, the user selects a particular problem report (by its title or initial contents), as shown in Figure 28. In this case the selected problem involves “Links that look like the buttons but don’t behave like buttons”.

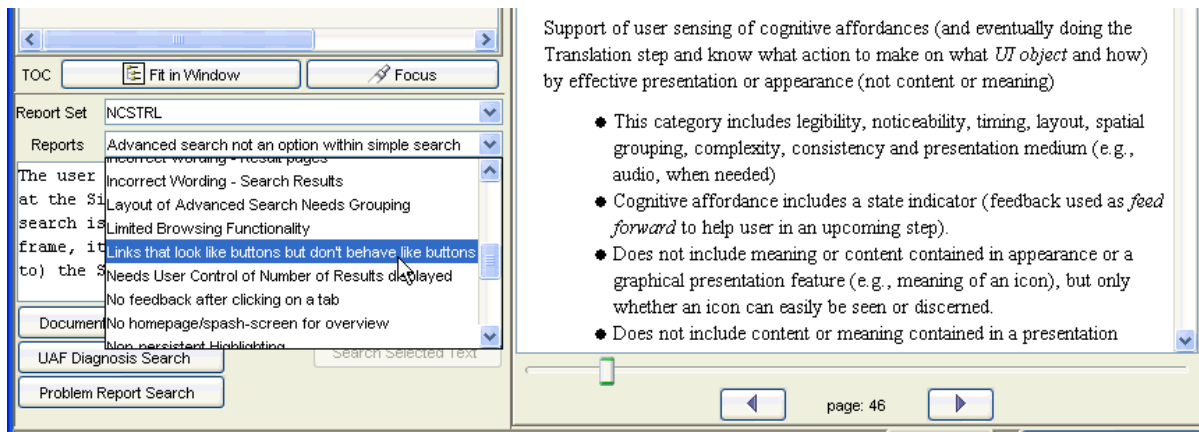


Figure 28. Selecting a Problem Report.

When a problem report is selected, the full text of that report is loaded from the database into the search query box. Here the user may read the problem report and, perhaps, edit it based on other materials (such as associated videos, screenshots, or other notes from the usability trial). Then, to begin the diagnosis process, the user may browse the UAF or use one of the search mechanisms to assist.

The most automated search mechanism to assist the diagnosis is the *UAF Diagnosis Search*. This search is invoked on the current contents of the query box when the user clicks on the UAF Diagnosis Search button, as shown in Figure 29.

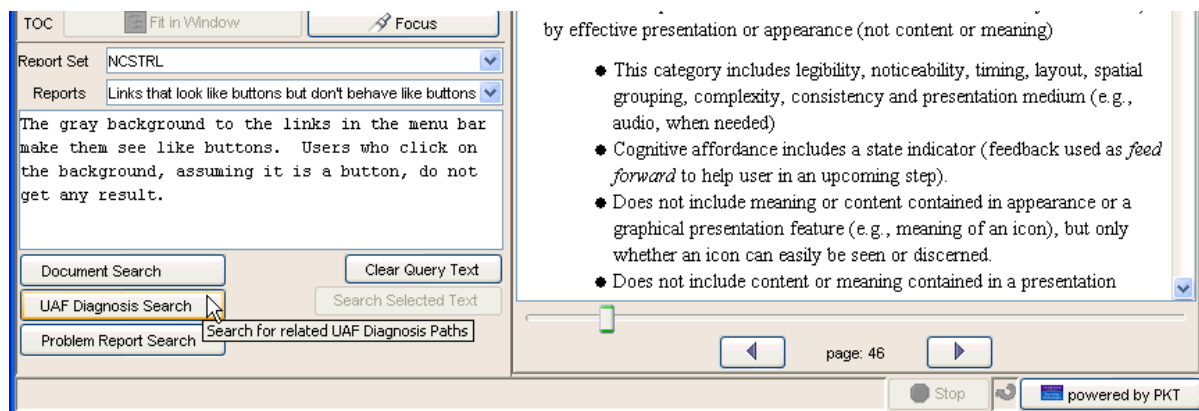


Figure 29. Invoking a UAF Diagnosis search on the text of a problem report.

The UAF Diagnosis search uses LSA to compare the text in the query to all possible diagnoses in the UAF, that is, to all paths through the UAF from the root node. The document vectors for these paths are pre-computed using the usability semantic space. These “path documents” are automatically generated from the UAF by concatenating the relevant text from each node along the path. Relevant text is defined as titles, descriptions, and examples. The contents of the cross-reference sections are not included, as these texts refer primarily to the concepts and definitions of other UAF nodes.

When the user clicks the “UAF Diagnosis” button, the contents of the search query box are sent as a query request message from the ST Browser to the Software Therapist Query Server. This communication occurs over a network (i.e., a Local Area Network (LAN) and/or the Internet) using a simple XML-based protocol.¹⁰ The query request contains the text of the query and specifies a query type: a UAF Diagnosis Search, in this case. The query server then processes the query. This involves converting the query into a vector representation appropriate for the semantic space, then comparing this vector to those for each of the possible UAF diagnosis paths in the UAF, and returning these results as an ordered list of the paths ranked by their degree of similarity to the query. These results are then sent as a query response message from the query server back to the ST Browser, which displays the results for the user. The results of the query submitted in Figure 29 above are shown in Figure 30.

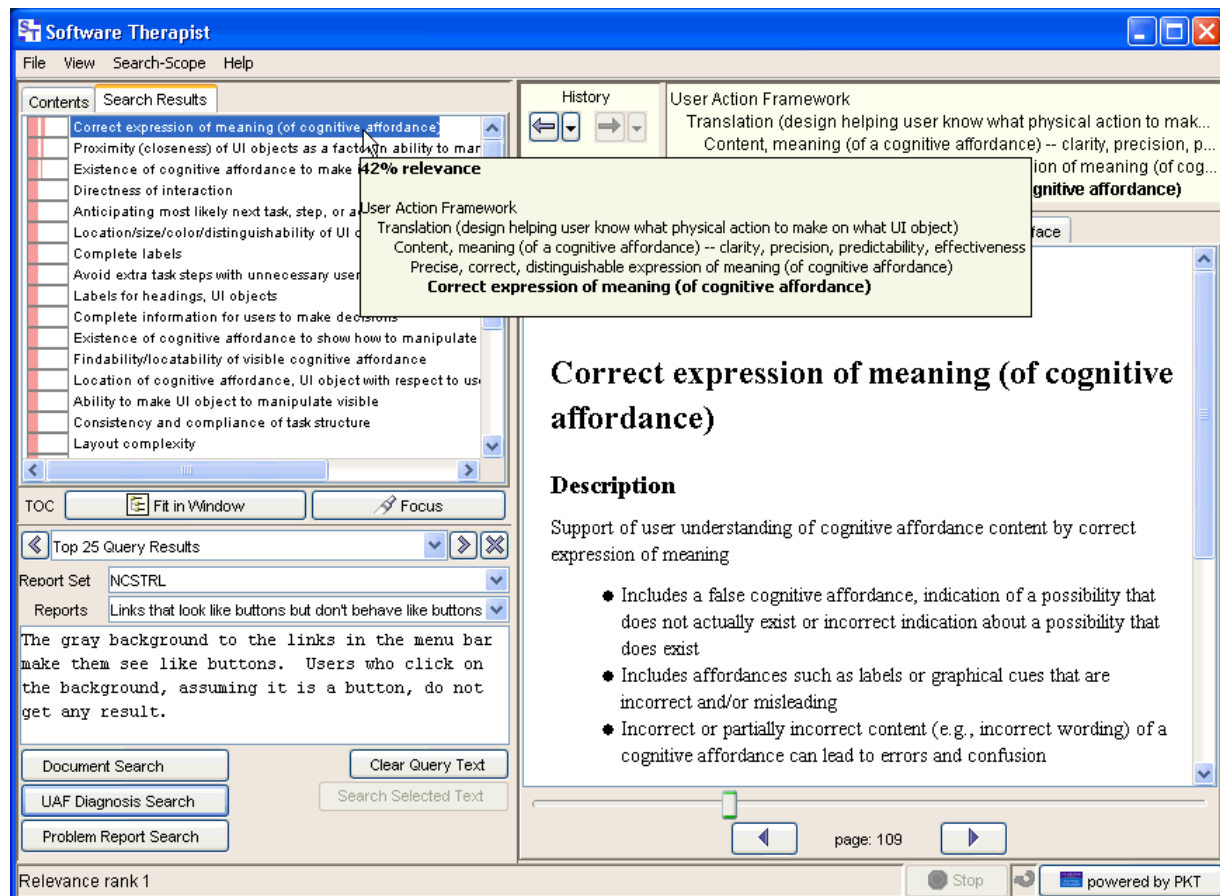


Figure 30. Ranked results displayed in the Search Results panel, showing a 42% relevance ranking for the top search result, a UAF diagnosis path terminating at the UAF node “Correct expression of meaning...”.

In this figure, the ranked list of results is shown in the “Search Results” panel at the top left of the ST Browser. This panel displays search results in order from most to least relevant (similar) to

¹⁰ XML (for *Extensible Markup Language*), is a widely used standard for network-based services. See, for example., <http://www.w3.org/XML/>, <http://www.xml.com/>.

the query. A “thermometer” graphic gives a rough visual indication of the degree of similarity, and further details are shown when the user moves the mouse pointer over a particular result item. In Figure 30, the user has rolled the pointer over the top-ranked result, a UAF diagnosis path through the nodes Translation, etc., and ending at the terminal node “Correct Expression of meaning (of cognitive affordance)”, as shown in the tool tip. The contents of the selected final node of the top result path are displayed in the Main Display panel on the right.

Figure 31 shows another view of the same search results. Here the user has clicked the TOC (“Contents”) tab at the top left. This displays the search results in the TOC, one result at a time in its context within the UAF. The figure shows the top search result. The TOC is opened to show the path representing this diagnosis, with each node in the path highlighted in light blue and the final node in dark blue. The path summary panel similarly displays each node in this analysis path. This view is typically a more useful overview of the analysis, e.g., when the TOC view has had to scroll part of the path out of view in order to display the terminal node in the path. The “Focus” and “Fit in Window” buttons can also be used to collapse the TOC display while maintaining visibility of the currently selected node.

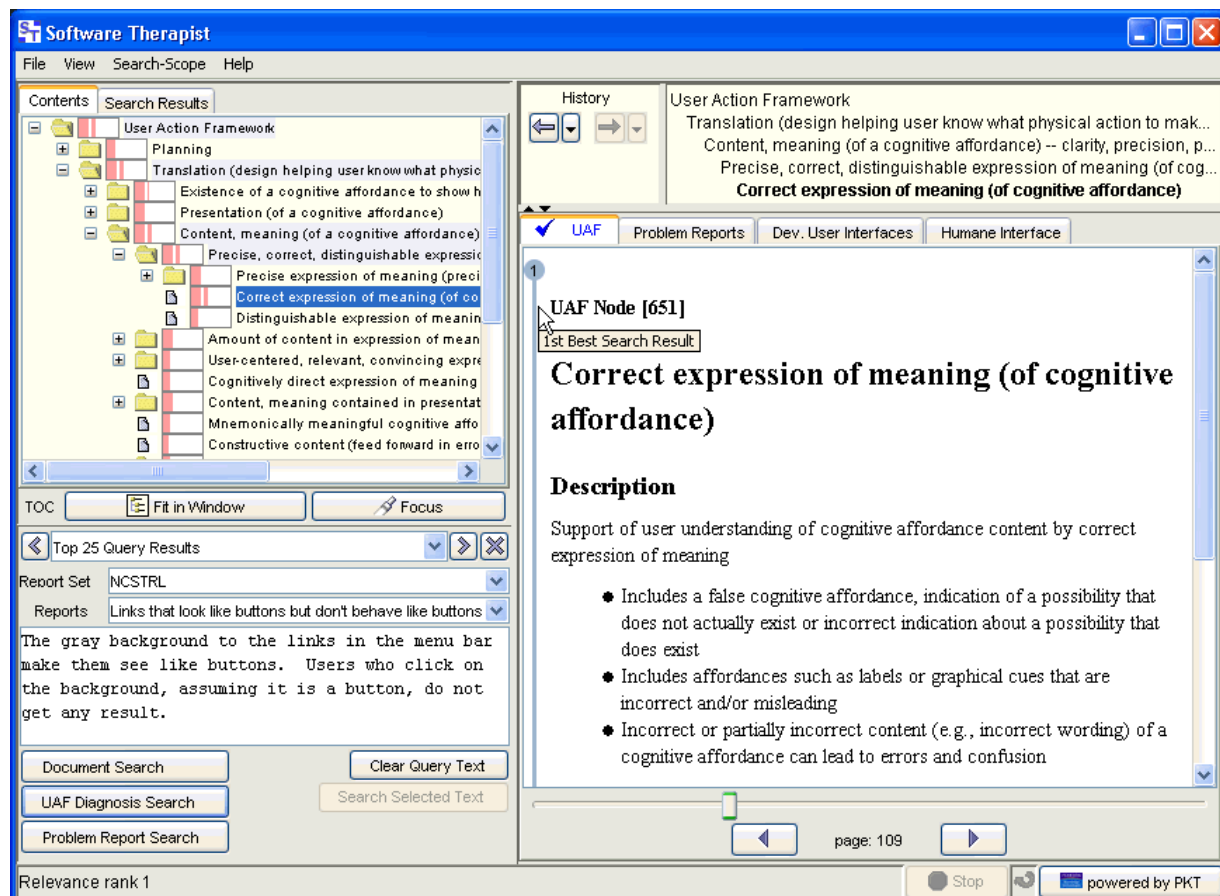


Figure 31 UAF Diagnosis search results showing the top search result in the TOC, the path summary view, and its contents in the main display panel.

The full contents of the selected final node in this analysis path appear in the Main Display panel. That this is the top search result is indicated by the blue bar in the left margin with a “1” at the top. Rolling over the margin also shows a tool tip indicating “1st Best Search Result”.

The ST Browser provides several ways to browse the search results. From the Search Results panel, the user can quickly scan the ranked list of results. This is shown in Figure 32, where the user has rolled the pointer over a result showing “28% relevance” that ends at the node “Location/size/color/distinguishability of UI object...”. Clicking here in the Search Results display navigates to the targeted node, producing the view shown in Figure 34.

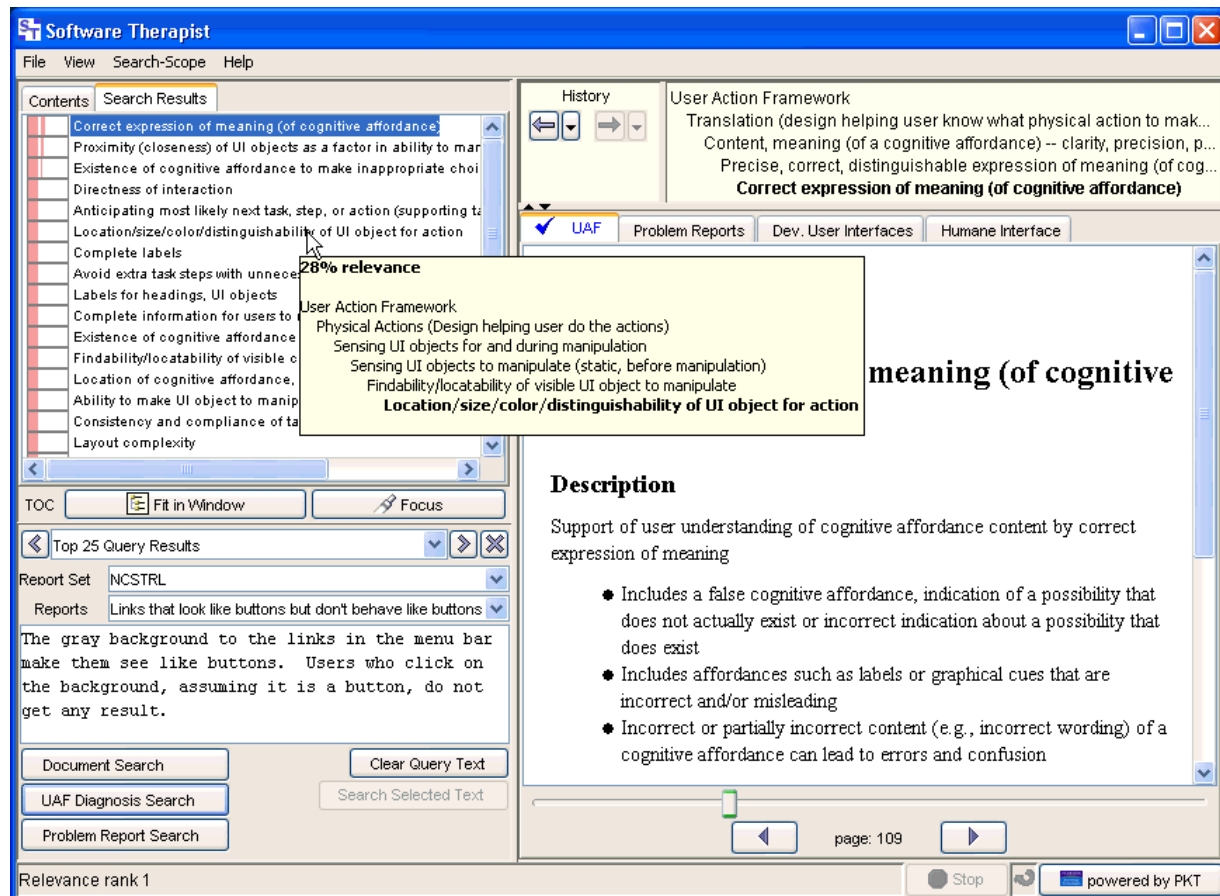


Figure 32. Ranked search results with rollover tool tip showing degree of relevance/similarity and overview of result diagnosis path.

In Figure 33 the user has navigated to the 6th-best search result. Another way to jump to a different search result is with the Top Query Results pop-up list. In Figure 34, the user is about to jump to the 13th result using this feature.

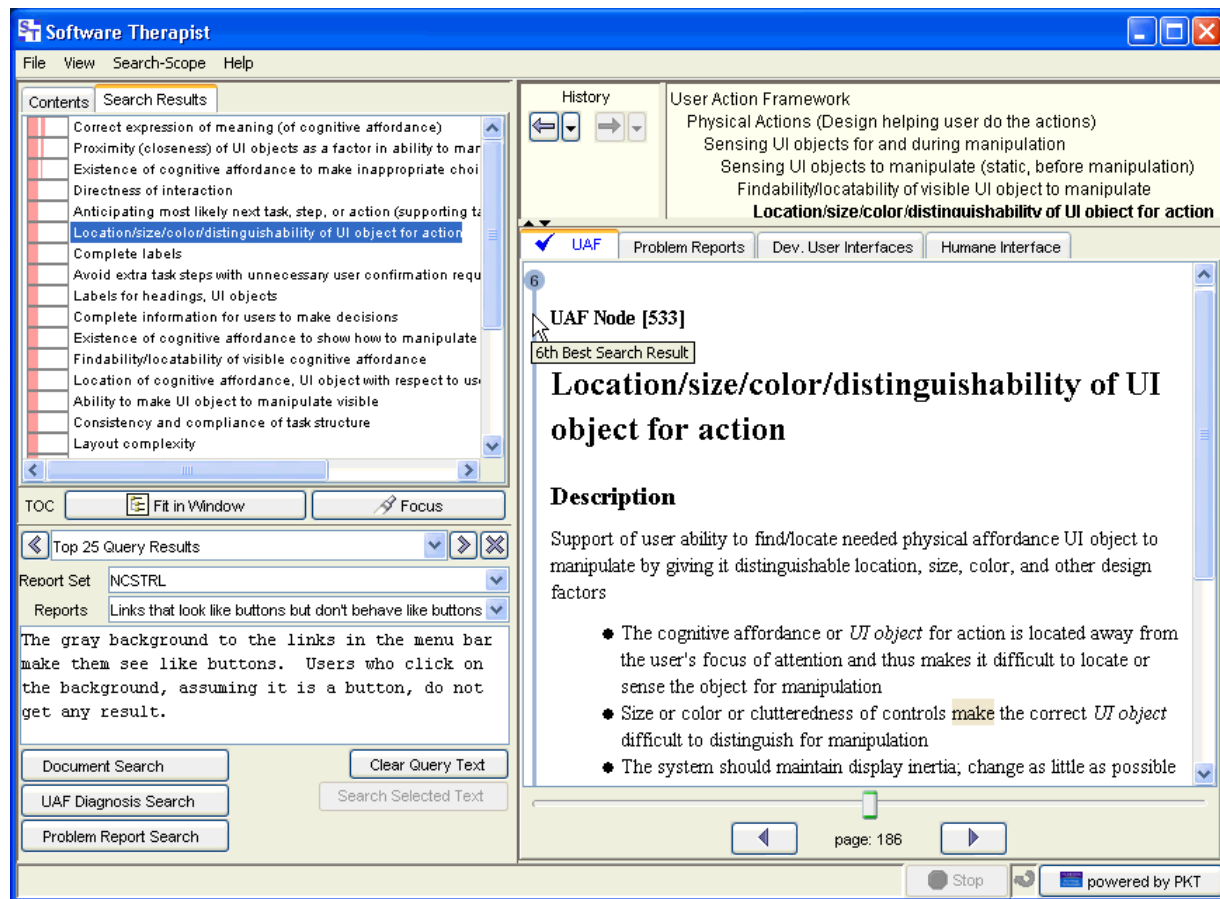
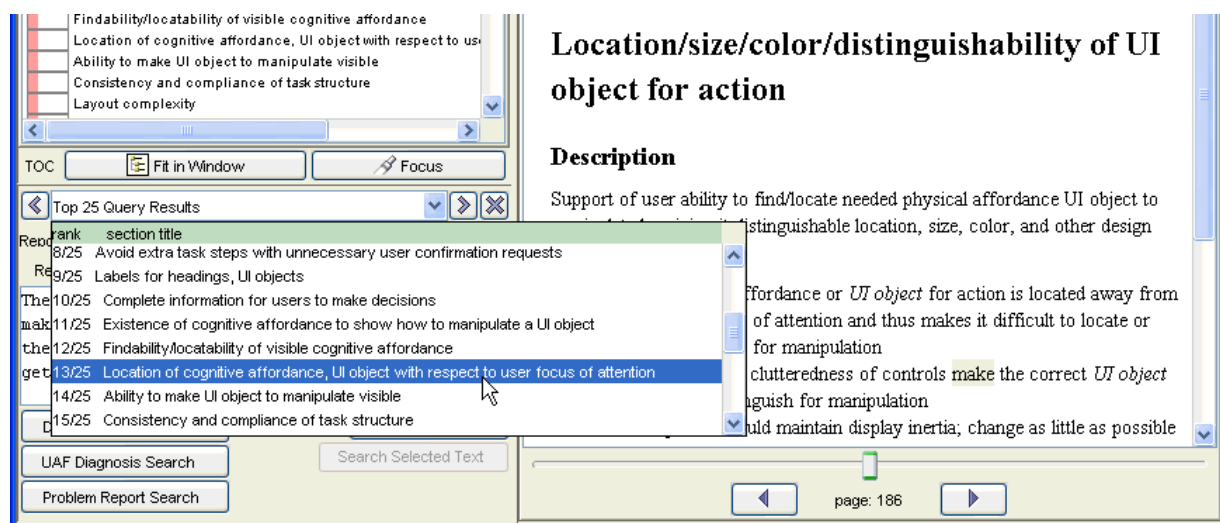
Figure 33. The 6th-best search result.

Figure 34. Using the Top Query Results pop-up box to navigate to another search result.

After examining and evaluating several possible diagnoses, the user may have selected which diagnosis is correct. However, if the entire diagnosis is not certain, at least a partial diagnosis may have been arrived at. A partial diagnosis may, in fact, already be available when the diagnosis process begins, for example if the first level of diagnosis (i.e., immediate intention as Planning vs. Translation, etc.) has been determined definitively during the usability trial or with the Diagnosis Wizard. In these cases, the user may still want to make use of LSA-based search for candidates for the final diagnosis, but with the scope of the search limited to those sub-branches of the UAF that reflect any partial diagnosis that may already have been decided upon.

This type of search scope limitation is supported in the ST Browser. An example is shown in Figure 35. Here, the user has activated the search scope feature using the “Search Scope” menu, which displays a checkbox associated with each UAF node. This allows the user to click on the checkbox for any combination of nodes to be included in a subsequent search. Selecting the search scope checkbox for a given node has the effect of including the entire branch of the UAF that is rooted at that node. That is, the search will include that node and all its child nodes, their children, etc. This can be changed by opening that node and unselecting any sub-branches that should be excluded from the search.

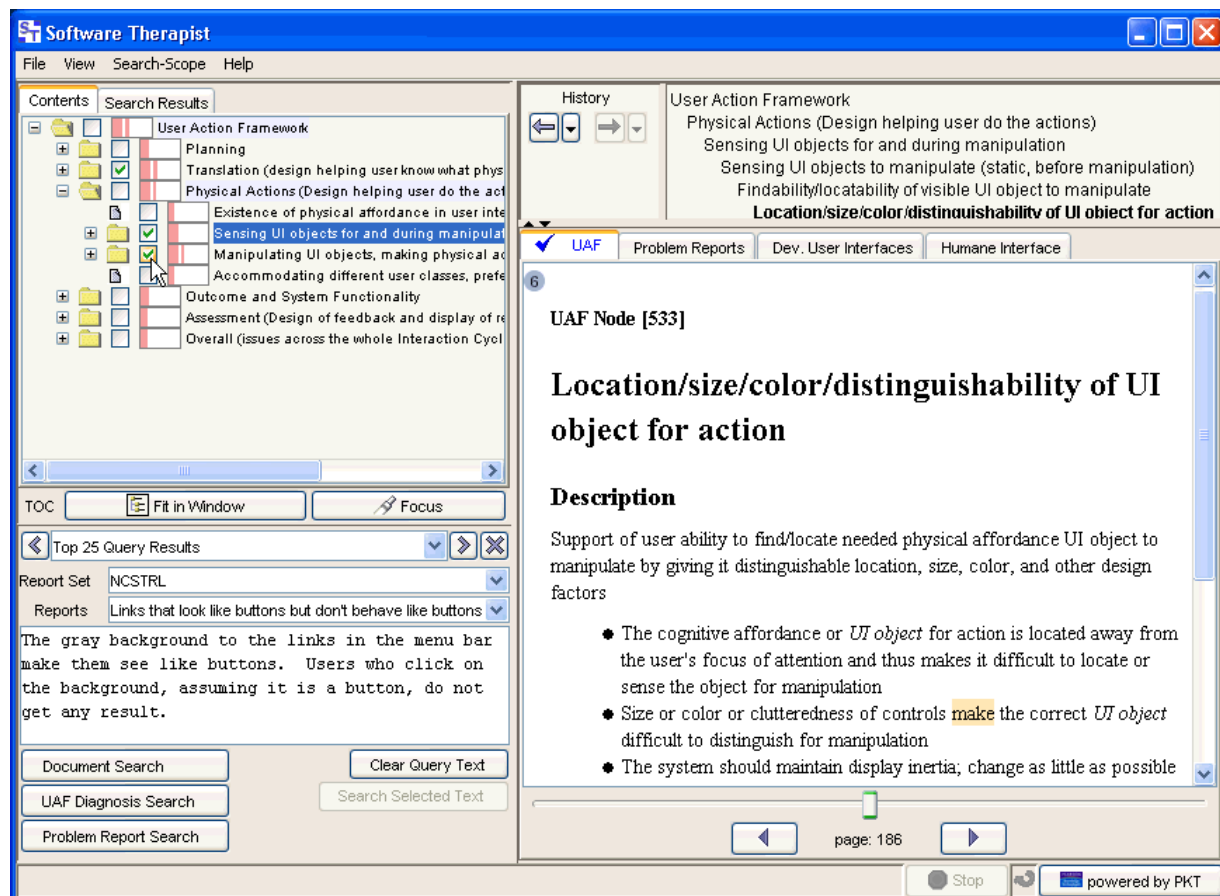


Figure 35. Limiting search scope for follow-up search.

An example of search scope control is shown in Figure 35. Here, the user has enabled search scoping and has selected checkboxes to limit a subsequent search to three sub-branches of the UAF: the “Translation” branch and the two sub-branches under “Planning” rooted at the nodes “Sensing UI objects...” and “Manipulating UI objects...”.

The results of executing this scoped search are shown in Figure 36, where the TOC has opened to show the path to the terminal node for the top-ranked result, the contents of which are displayed in the Main Display panel.

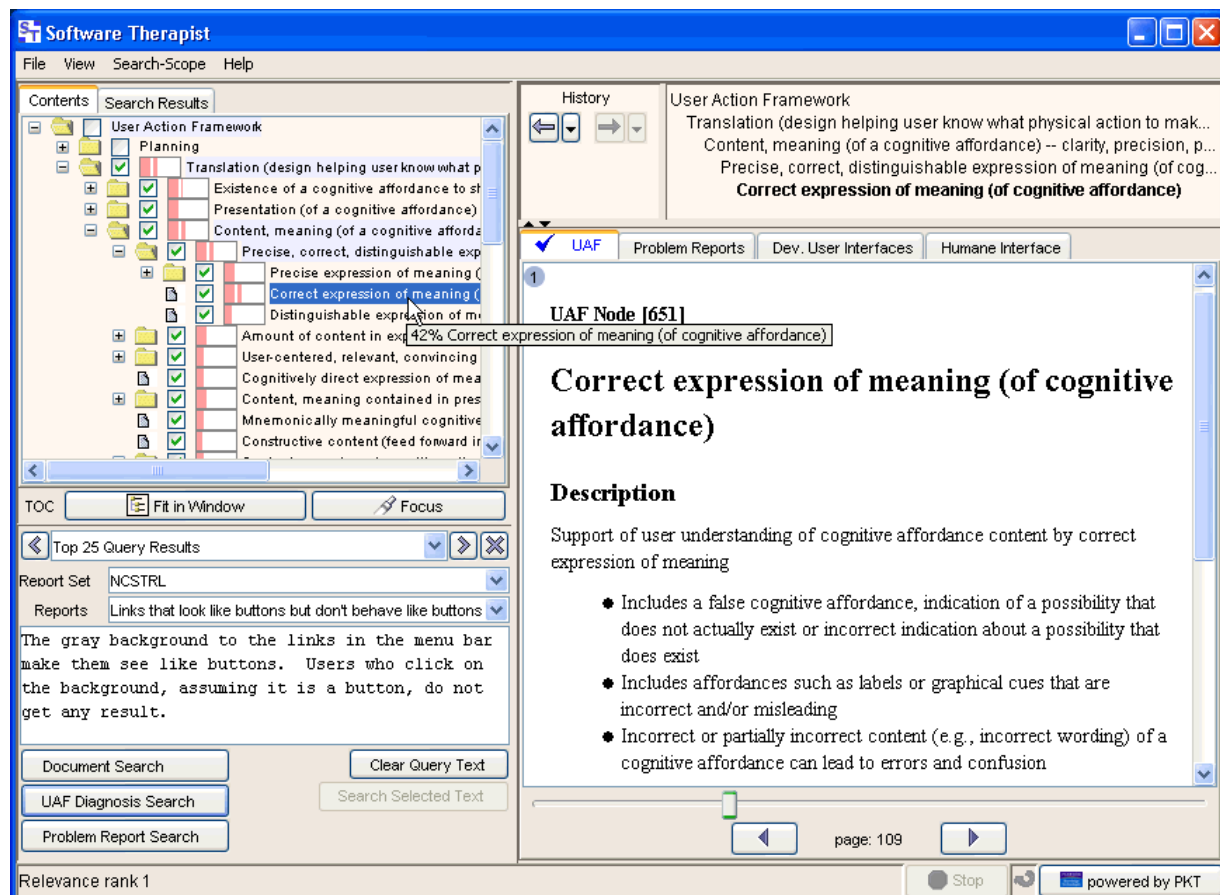


Figure 36. Results of the scoped search.

After finishing the diagnosis for one problem report, the user may want to take a break to get a cup of coffee or take a short walk, but when ready to continue with another problem report, he or she can clear the ST Browser's current search state by clicking on the “X” at the top right of the search panel, to the right of the “Top 25 Query Results” pop-up selector (as in Figure 36, for example.) This resets the display, removing the top results selector and the relevance “thermometer” icons, clearing the query text box, etc. At this point, the user can select another problem report from the database or, as shown in Figure 37, enter free text into the query text box. In Figure 38, the user has selected the 7th-best result from a UAF Diagnosis search on this text as the correct diagnosis.

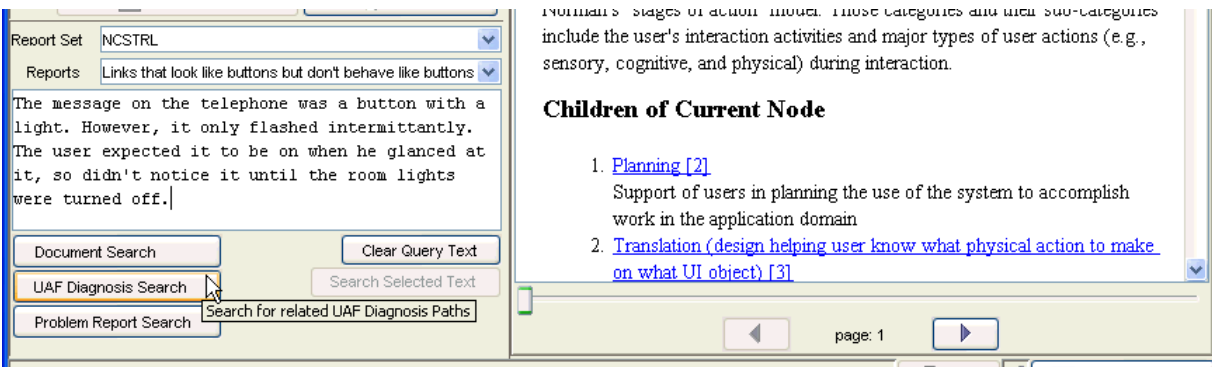


Figure 37. Entering the text for a new problem description into the query box, then invoking a UAF Diagnosis search.

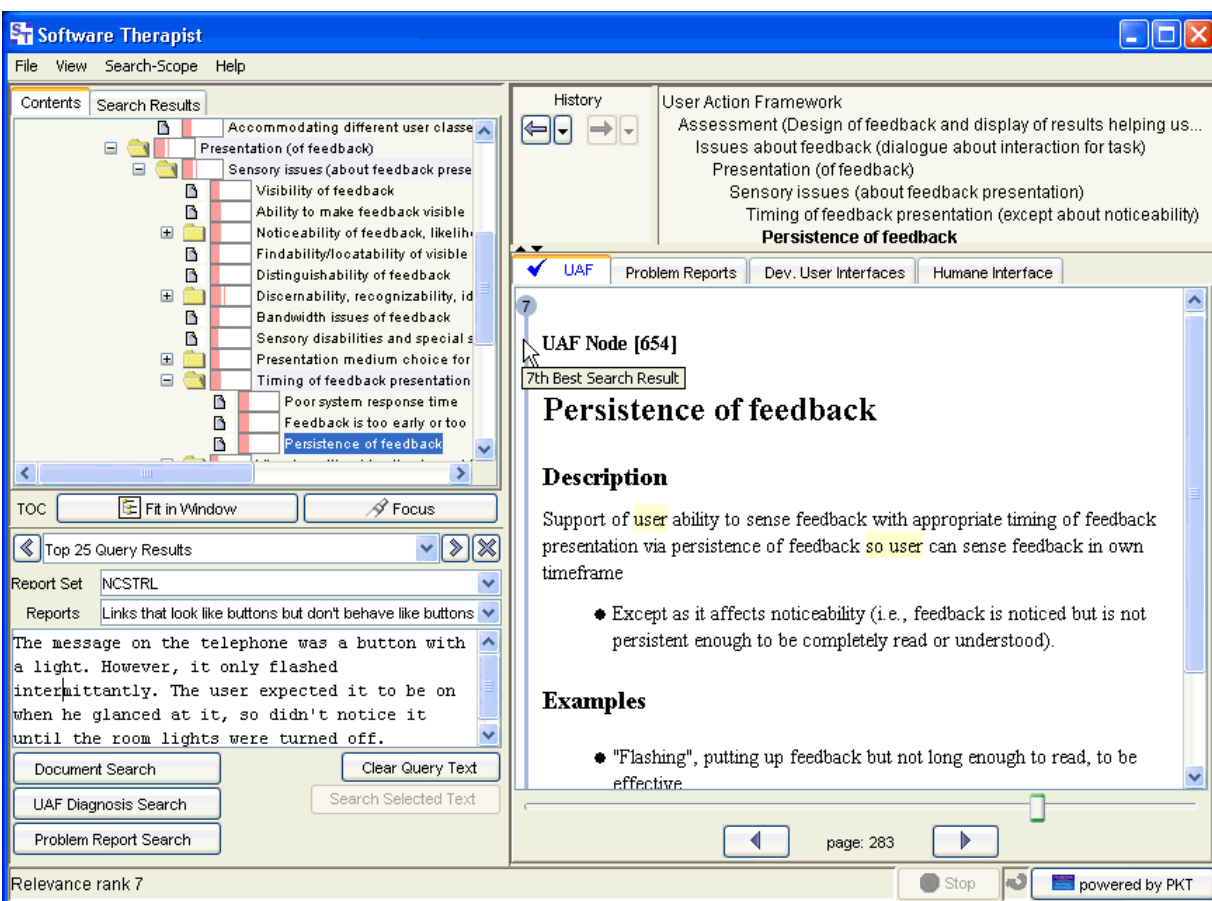


Figure 38. The ST Browser after the user has selected the 7th-best result of a UAF Diagnosis search on the new query text.

As part of the problem diagnosis process, the user may want to compare the current problem report text to other problem reports in the database that may have involved similar circumstances and issues. The ST Browser supports this with its LSA-based “Problem Report Search” feature,

which is always available in the ST Browser. For example, in Figure 39 the user is browsing the UAF while examining a problem report from the “MISC” problem report set titled “Users need an indication of which objects are clickable”. The figure shows the user about to click the “Problem Report Search” button.

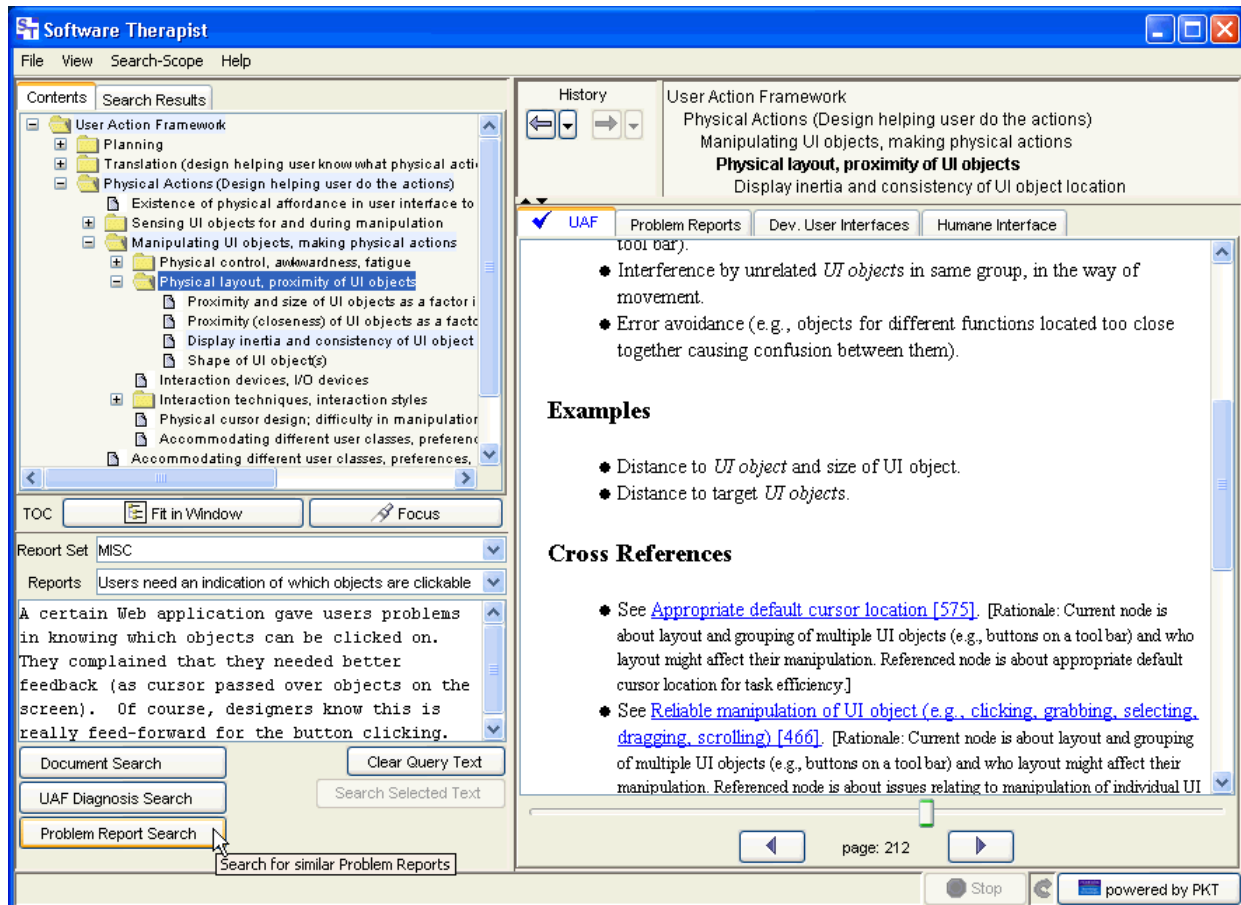


Figure 39. Invoking a Problem Report Search from the UAF.

The result of the search invoked in Figure 39 is shown in Figure 40. The problem reports in the database have been ranked by their semantic similarity to the query text, and these results are shown in the Search Results tab at the top left. The user has selected the 2nd item here, which shows a 38% LSA-determined relevance to the query. The contents of this report are shown in the Main Display area on the right, which has moved to the “Problem Reports” tab.

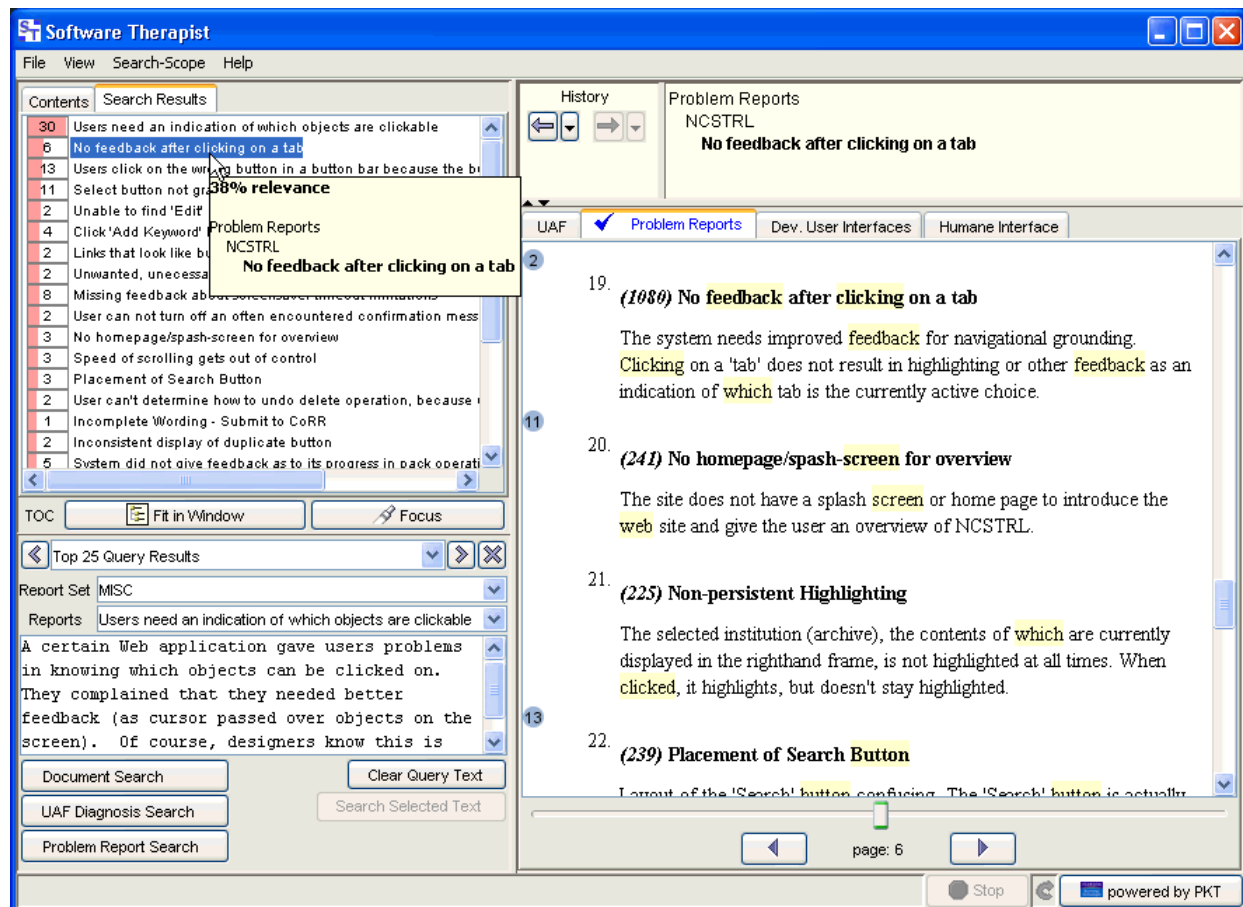


Figure 40. The results of the Problem Report Search invoked from the UAF

The problem reports available in the database can be browsed as well as search in this display, using the TOC and other navigation tools in the ST Browser, as shown in .

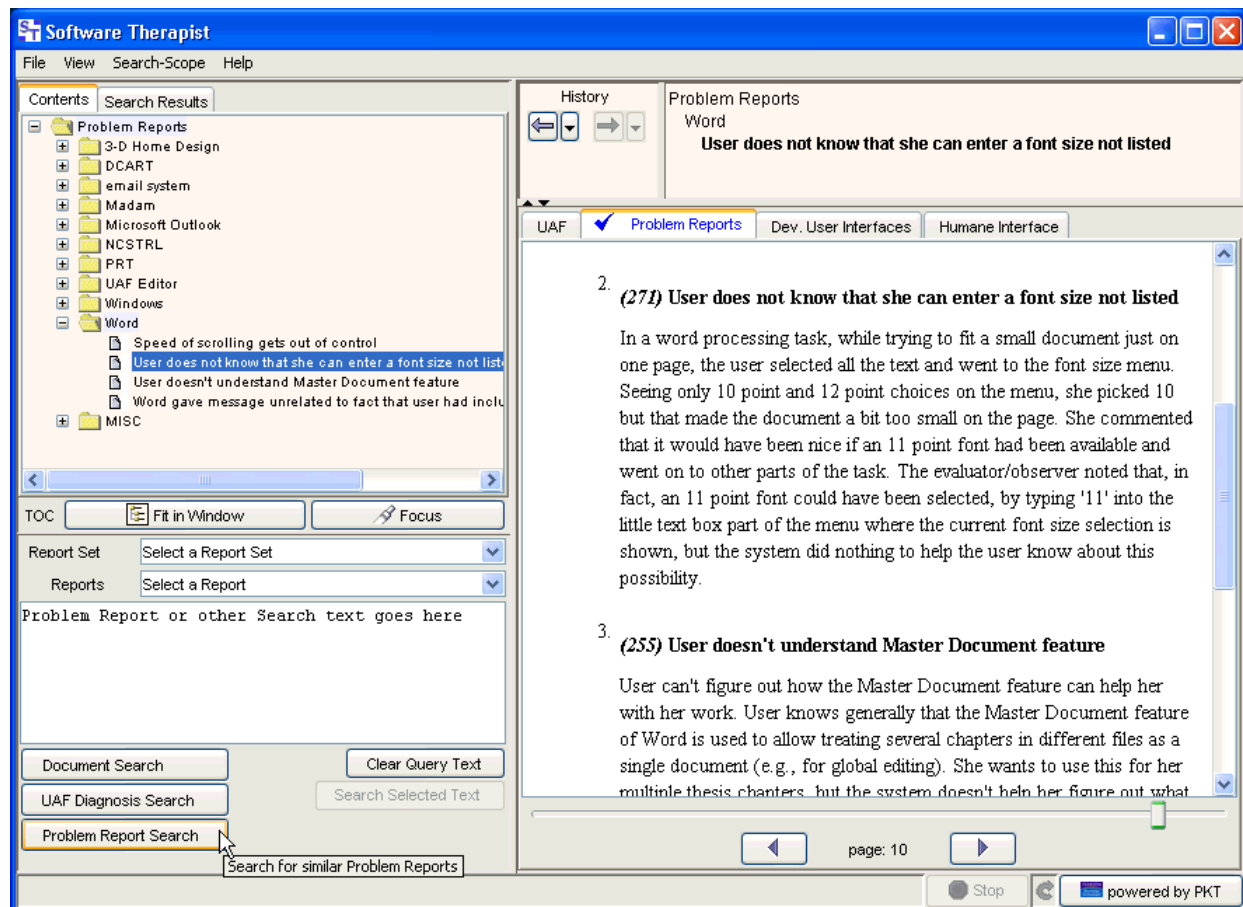


Figure 41. Browsing the Problem Reports in the ST Browser.

2.2.3.3 Using LSA to search for related literature

As part of analyzing a problem report, or perhaps for other reasons, users may want to examine resources other than the UAF. The ST Browser can make these available in the form of related documents that have been imported into the Software Browser system. The import process involves converting the source document to HTML¹¹ (if they are not already in that form), segmenting them into pages and passages (as “documents” for LSA search purposes), generating text vectors for the resulting documents, and deploying these resources to the Software Therapist server, which makes them available to users of the ST Browser as one of several library items, as shown above in Figure 21.

¹¹ For HTML (*HyperText Markup Language*), see, for example, <http://www.w3.org/MarkUp/>.

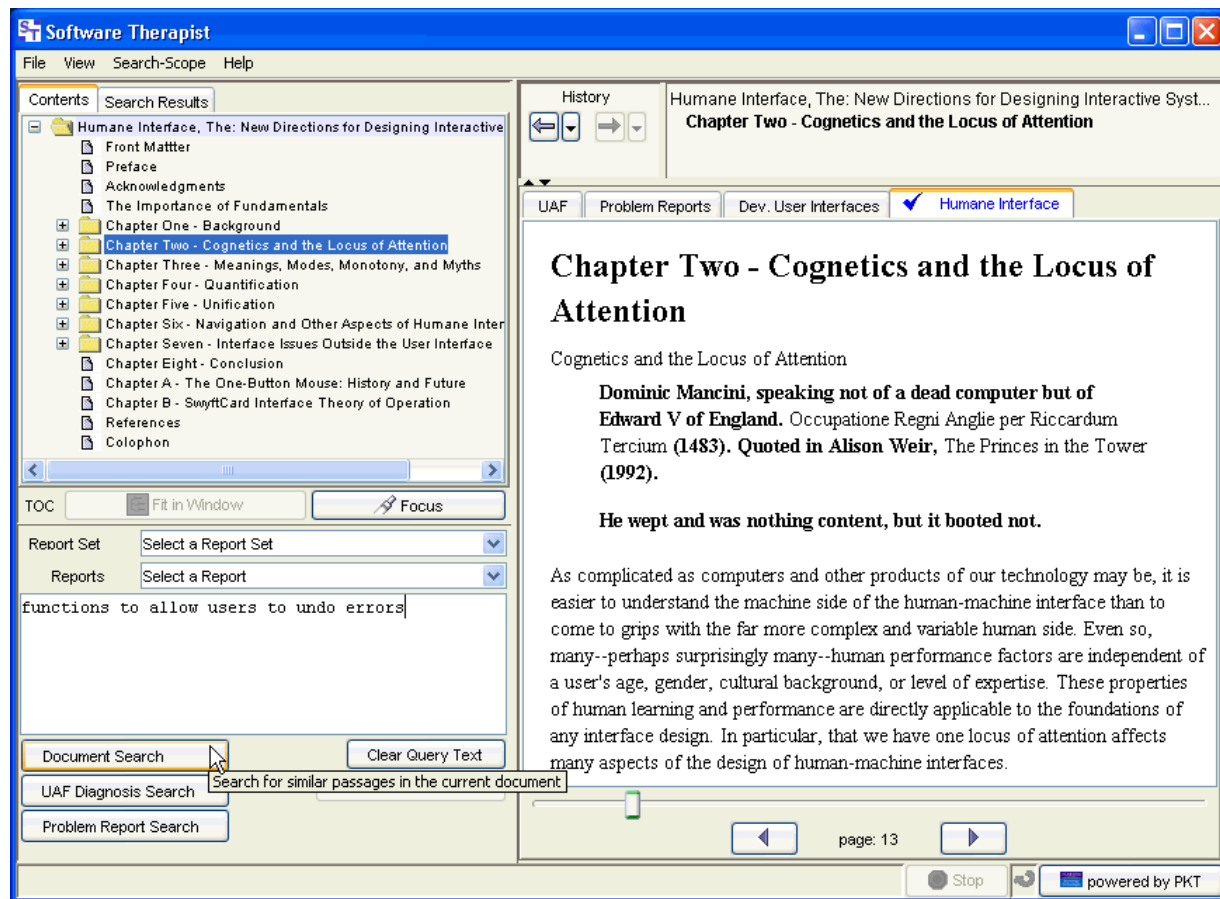


Figure 42. Searching an online textbook for semantically similar sections using Document Search.

After selecting and loading a literature resource from the library, it is displayed in a new tab in the ST Browser along with the same controls available when browsing the UAF, as shown in the example in Figure 42. The TOC shows the structure of the document as an active tree structure, allowing the user to jump to selected sections of the document; page controls allow the user to read page-by-page, as the search panel is available to provide LSA-based search of the current document.

In Figure 42, the user, while browsing page 13 in chapter two of the displayed book, has typed the query “functions to allow users to undo errors” into the query box and is about to press the “Search” button. This will invoke a basic LSA search for the query against the currently displayed document, returning the passages in the document in order of similarity to the query. The results of this query are shown in Figure 43, where the user has browsed to the 3rd-most similar passage to the query.

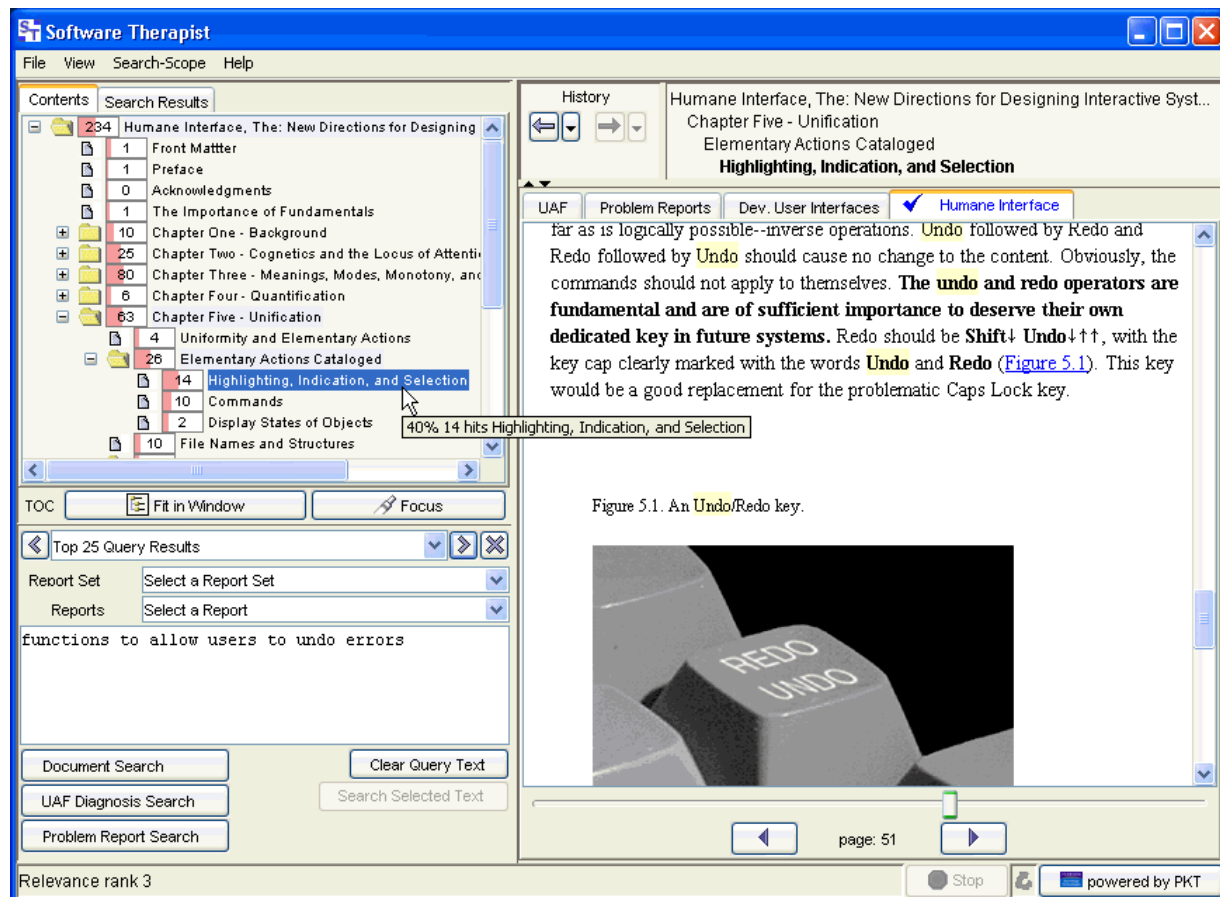


Figure 43. Search results.

The TOC shows the search results by highlighting the location of the currently displayed result. In addition, it shows the familiar “thermometer” icon, indicating the degree of semantic similarity to the query. The thermometer icon also shows the number of “keyword hits”, i.e., words from the query that occur verbatim in the passage found. Rolling over the TOC item displays this information in a tool tip. Keyword hits are also highlighted in yellow in the main content display.

The LSA-based document search mechanism assists the user in investigating a topic in related literature, typically as part of the usability problem analysis process. This type of search may also be invoked on an arbitrary selection of text, as shown in .

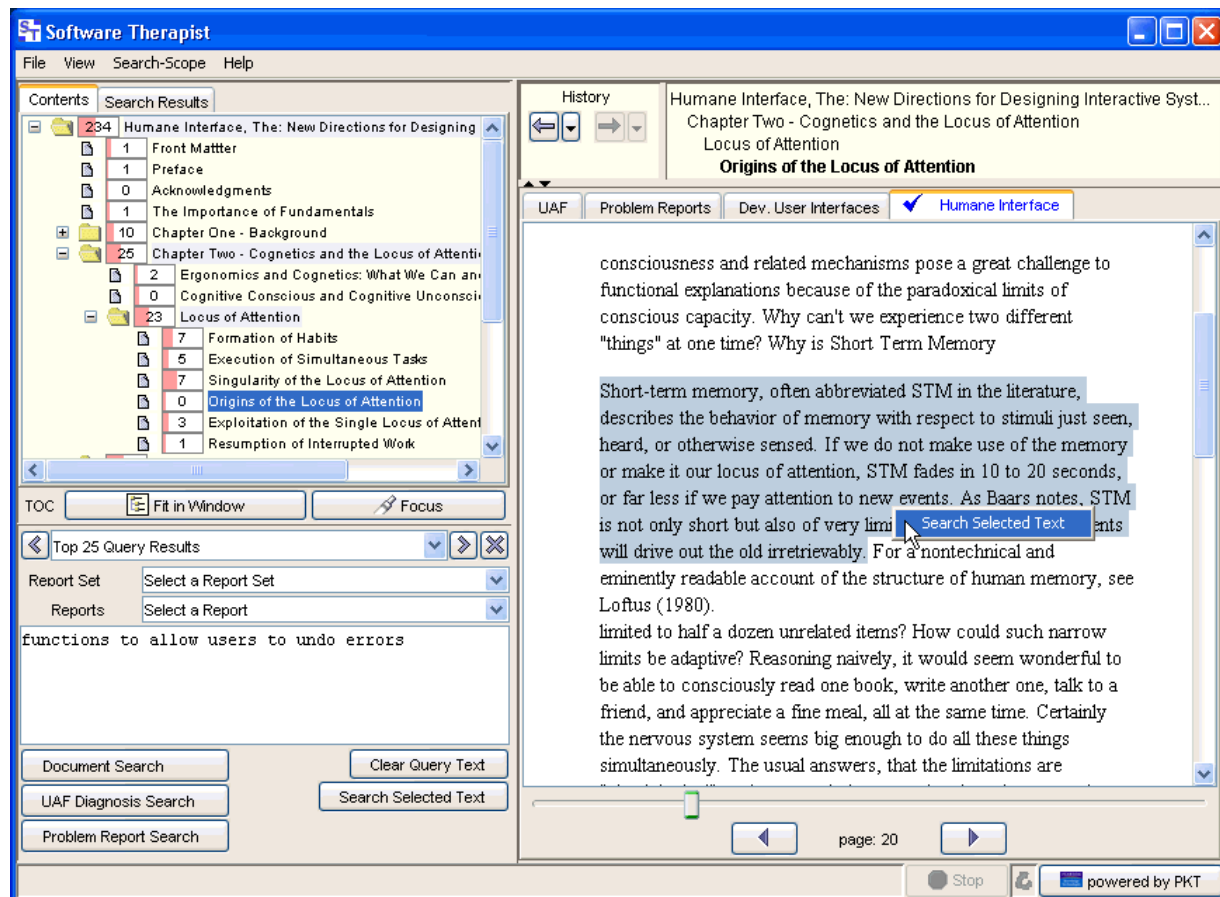


Figure 44. Invoking Document Search on a portion of text selected with the mouse.

At the same time, the “UAF Diagnosis Search” feature remains available. That is, at any point while browsing other literature, the user may invoke a search for UAF diagnosis paths that are similar to whatever text is currently entered into the query box.

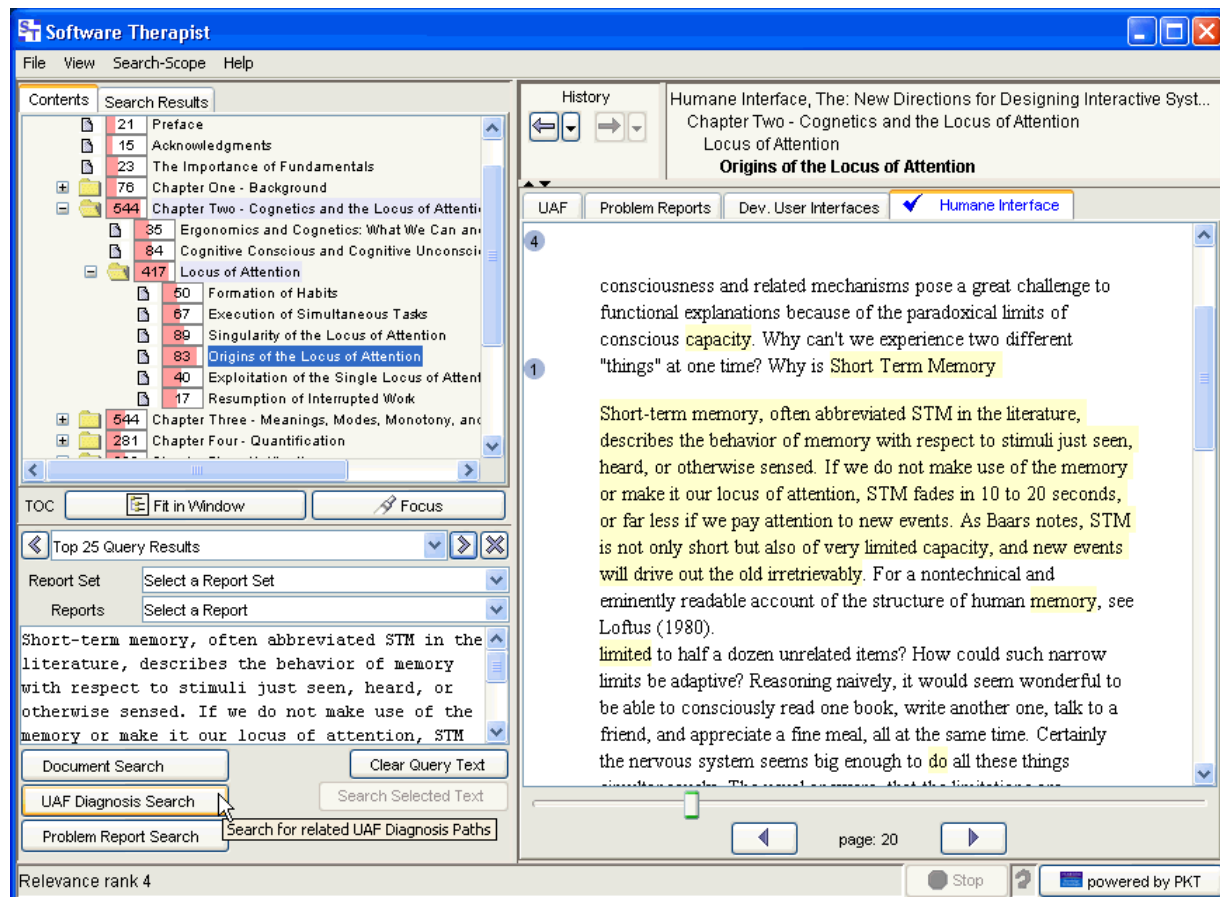


Figure 45. Invoking a UAF Diagnosis Search on text from a related document, pasted into the query box.

For example, in Figure 45 the user has pasted some text from a section of the current document discussing short-term memory into the query box and is about to invoke a UAF Diagnosis Search. The results of this query are shown in Figure 46. The TOC has opened the UAF and displays the top-ranked UAF diagnosis path, which is also displayed in the path summary panel (top right). The document display panel has switched to the UAF tab and displays the content of the final node of this path, titled “Support human memory limitations”.

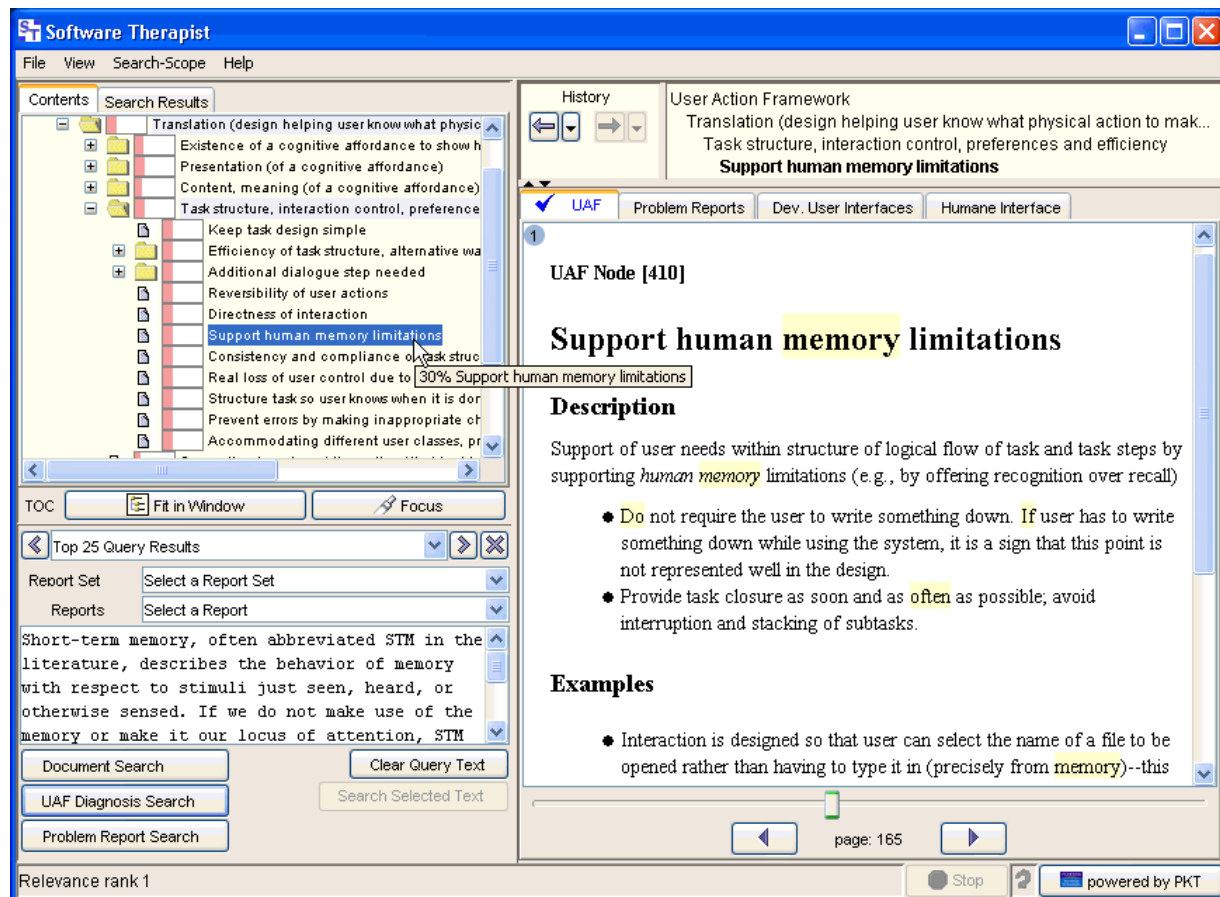


Figure 46. Results of UAF Diagnosis Search on text from a related document, pasted into the query box.

The query text that the user is working with remains in the query box for further modification and search operations. In Figure 47, the user has loaded a second document from the related literature library (in a new tab in the main document display) and has conducted a document search for the same query within this new document. The top search result is a section titled “Human Memory Issues” with “42% relevance” to the query.

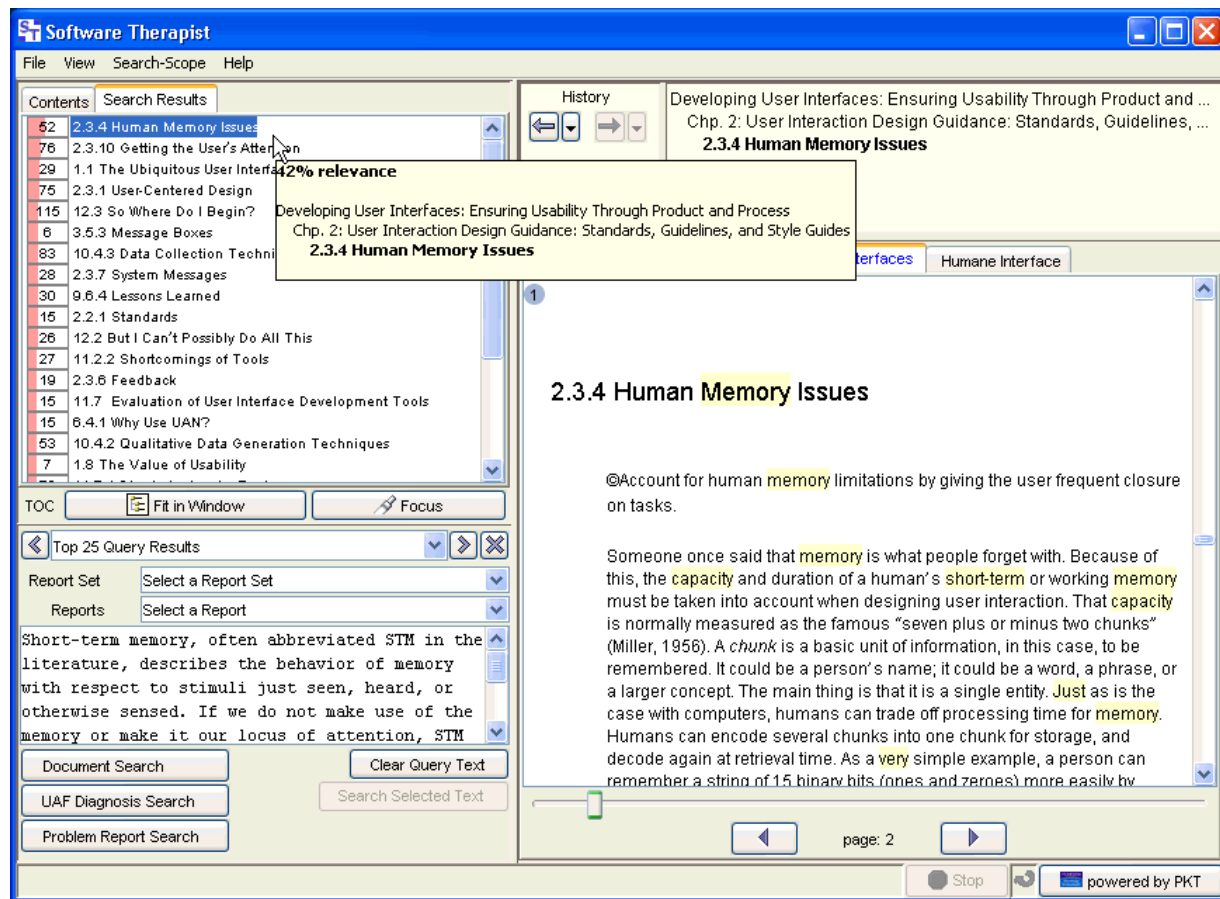


Figure 47. Results of a Document Search on the same text in another related-literature resource. Note that the degree of relevance (LSA-based similarity) is not simply a matter of the most (or greatest density of) keyword hits, as suggested in the (LSA-based) ranking of search results: The top-ranked result has fewer keyword hits than several other sections.

3 Research on the Application of LSA to Usability Engineering

3.1 Validation of UAF Content & Structure

During phase II we completed two major revisions of the UAF. For each revision, we reviewed and updated the UAF's structure and content. We changed structure in places where, for example, new terminal nodes were added to accommodate new example usability problems that were sufficiently distinct from previous examples to warrant a separate diagnosis. We changed wording for clarity and consistency, and to add distinguishers to better differentiate among different but similar diagnosis paths. (See below.) We also added examples and additional description for completeness and distinctness of UAF nodes

The updates were based on a number of inputs. For example, we frequently performed small-scale walkthroughs to ensure that the nodes were complete, consistent, and distinct. In addition, we updated the UAF to handle new cases encountered during the construction of the usability problem library. We also updated the UAF based on feedback from other academics who were using the UAF in research work and individuals who used it during field trials.

3.2 Role of “distinguishers” in Problem Diagnosis

A distinguisher is a wording (word, phrase, sentence, etc.) that represents the semantic differences between two UAF nodes or between a node and a set of other nodes. Distinguishers facilitate effective diagnosis decisions at any parent node by sharpening the clarity of the diagnostic choice among child nodes (in the process of finding a diagnosis path within the UAF). Distinguishers also play a leading role in the Wizard, where yes-or-no (or A-or-B) questions are posed to practitioners to direct them down correct UAF diagnosis paths.

Empirical derivation of distinguishers. Based on documented usability problems in our database and examples that establish the need for distinguishers based on cases where tool users have difficulty making a diagnostic decision based on existing node definitions, we have identified additional distinguishers and incorporated them into new versions of the UAF. We used LSA in two ways to support this process of refinement and extension of distinguishers in the UAF. First, we created a web-based semantic similarity tool for analysis of the UAF (an extension of existing KAT tools for LSA to the UAF and the usability engineering domain). This tool allow comparisons of the textual content of UAF nodes and individual words and phrases for degrees of semantic similarity, based on semantic spaces created from our collected library of HCI and usability engineering literature.

In addition, we created a web-based service that provides LSA-based semantic search of the UAF, described in more detail below. Access to this service is provided within the ST Browser. Using this search capability, UAF nodes can be browsed in the context of their similarity to other nodes or to arbitrary words and phrases that might serve as distinguishers between similar nodes. Expert users thus have an opportunity to identify distinguishers that would improve the UAF by better distinguishing nodes that address similar issues. This is one way in which the UAF may be edited and enhanced in the course of its use for usability engineering.

For example, Figure 48 shows the ST Browser viewing a node pertaining to the size of fonts in a user interface. Note that this node is at the end of a UAF path under the *assessment* branch of the

UAF. (This can be seen both in the highlighted nodes in the Table of Contents panel at the top left and the path summary at the top right.) We can also see that the user has right-clicked in the main content display panel for this node, bringing up a menu with links to the 10 most similar nodes. For each node in the UAF, the ST Browser has pre-computed the 10 other UAF nodes that are most similar, based on the semantic space for usability. In this way, the user can compare nodes in the UAF not only based on proximity in the UAF (e.g., by browsing with the TOC), but also based on semantic similarity.

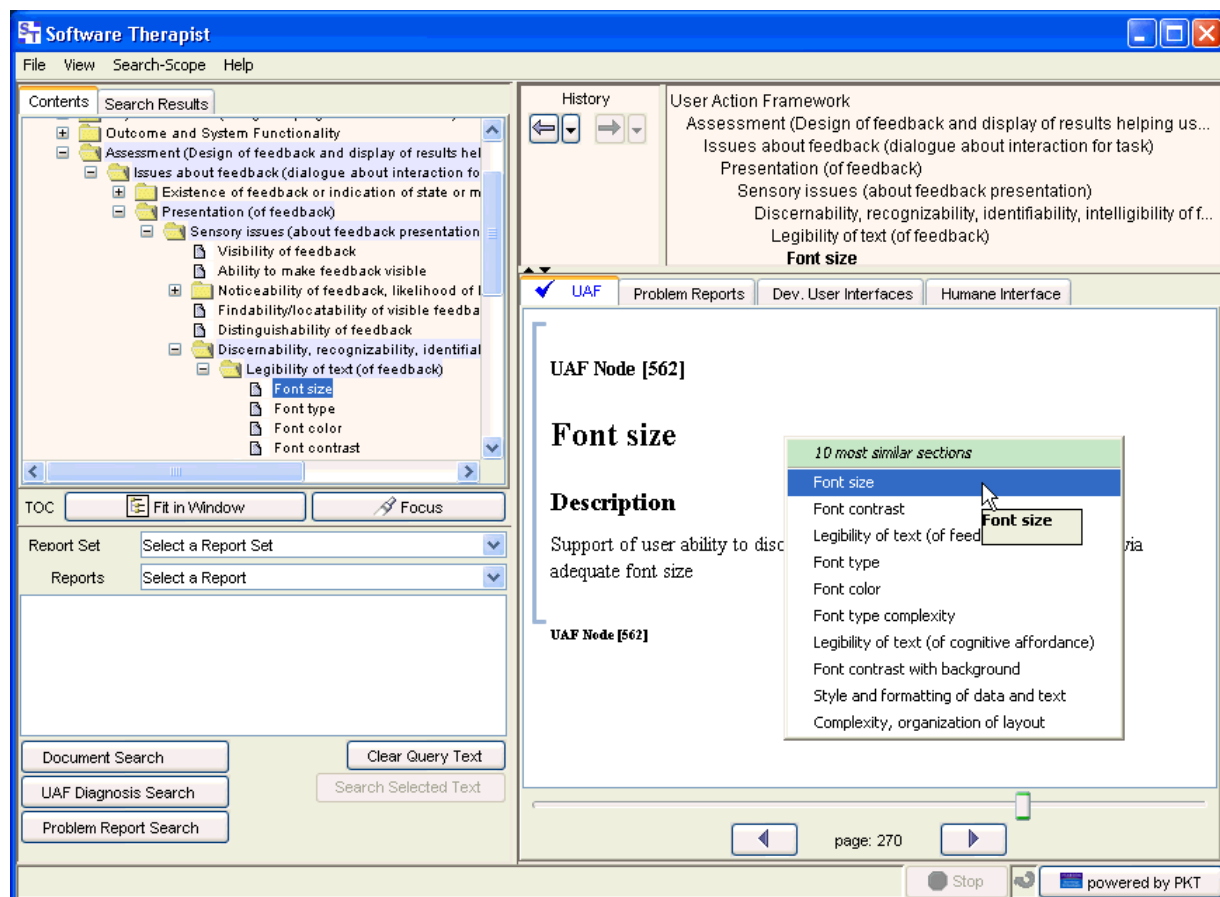


Figure 48. Using the pop-up context menu in the main display panel to show pre-computed links to the 10 most similar nodes in the UAF.

In this example, we see that semantically similar nodes pertain to topics such as font contrast, color, complexity, legibility, etc. In fact, the most similar node has the same title as the current node, “Font size”, although this node must lie down a different path. By following this link, the user arrives at this other “Font size” node, which is under the Translation branch of the UAF, as shown in Figure 49.

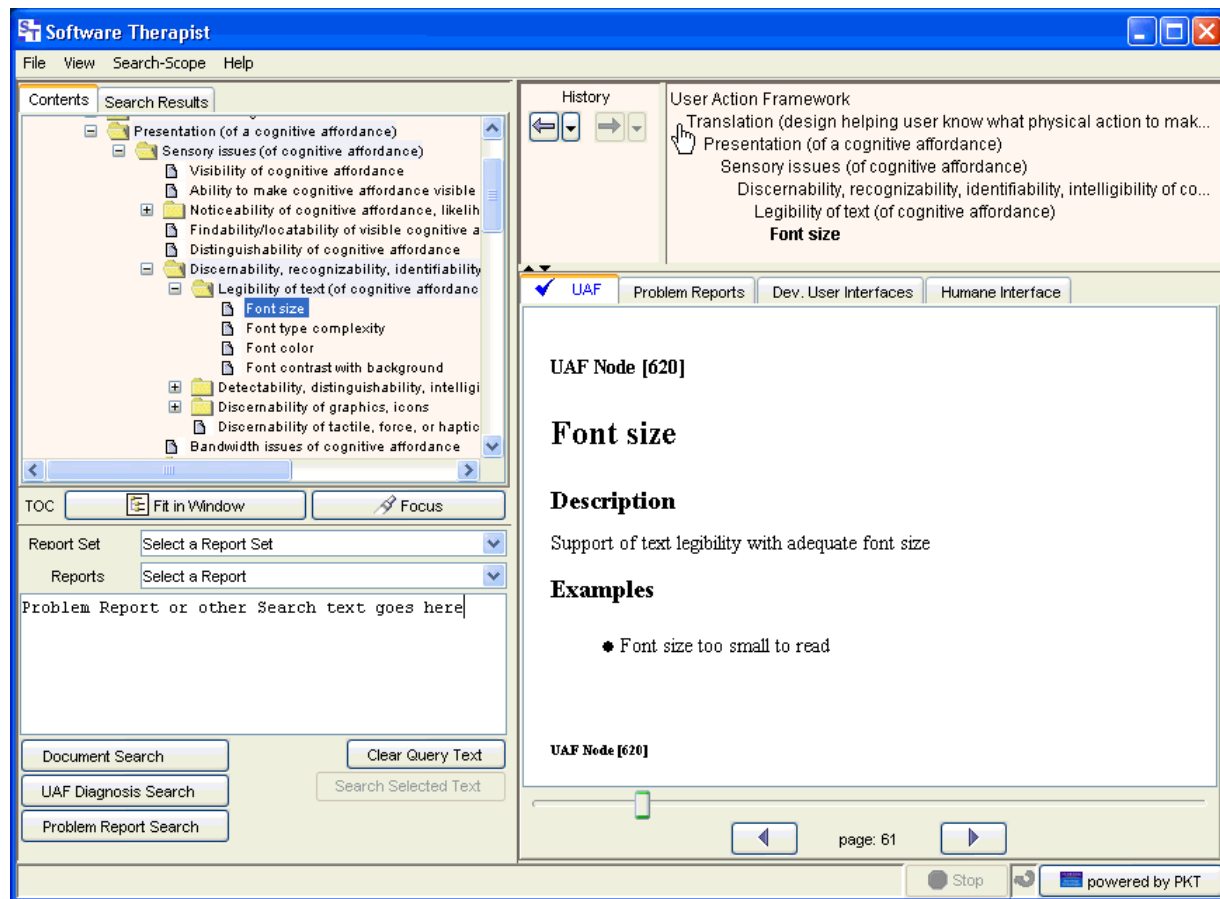


Figure 49. The UAF after the user has navigated to the node titled “Font Size” under the main *Translation* branch of the UAF.

Another way that users can search for similarities among UAF nodes is to select an arbitrary segment of text in the main display and right click to invoke the Search Selected Text function, as shown in Figure 50. In this way, a user analyzing the UAF can search for nodes most similar to a sub-section of a given node. The results of the search initiated in Figure 50 are shown in Figure 51.

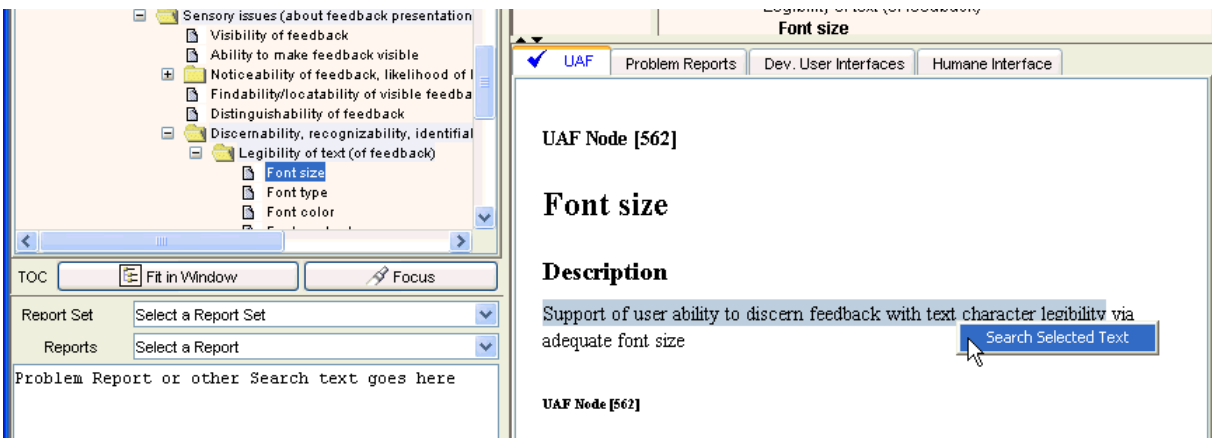


Figure 50. Invoking the *Search Selected Text* function from the main display of the ST Browser.

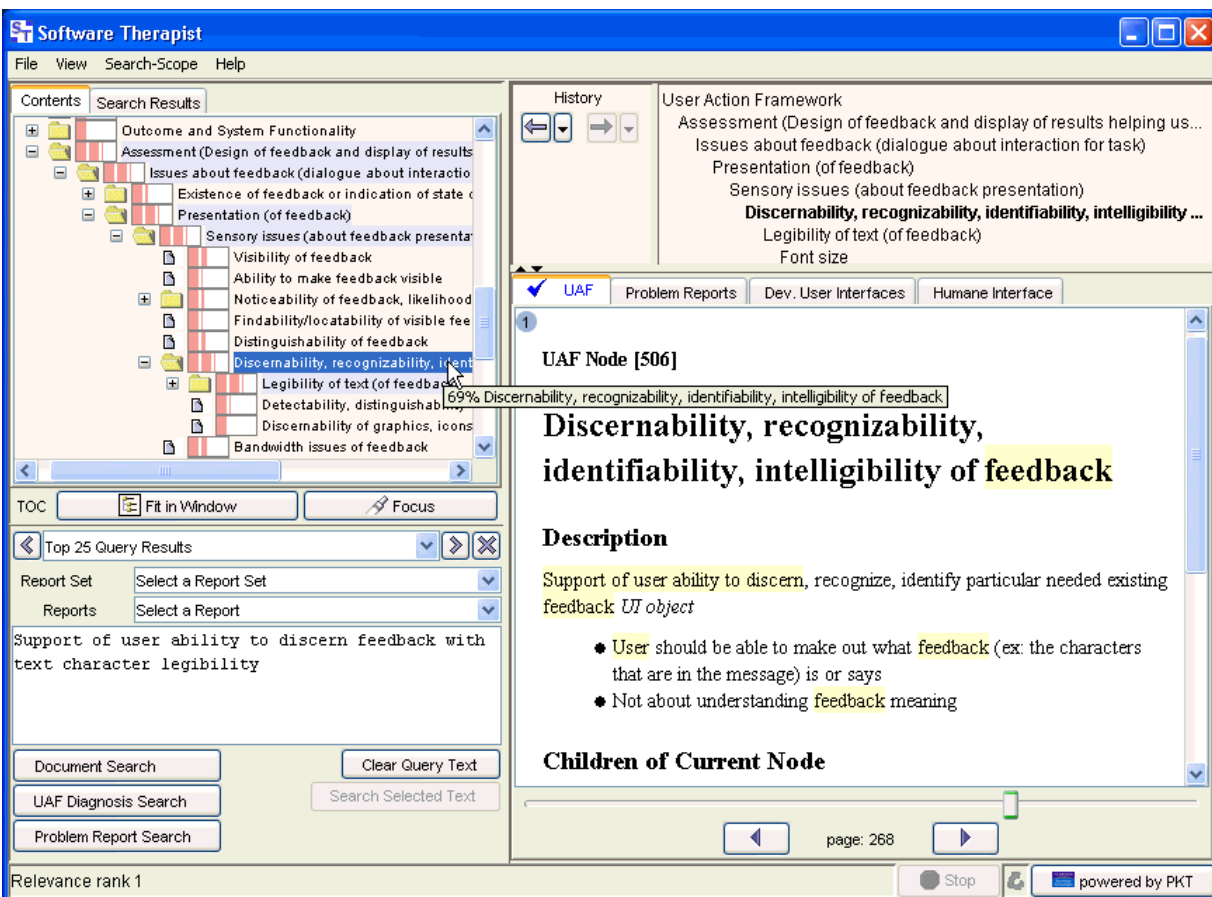


Figure 51. Result of the *Search Selected Text* function. The text selected as the query in Figure 50 is copied automatically to the query box (bottom left), the TOC is opened to the top search result and indicates the degree of similarity to the query, and the content of the top search results (UAF node 231) is loaded into the main display panel. Matching keywords from the query are highlighted.

Finally, a user analyzing the UAF can initiate a search for which nodes are most similar to any arbitrary text entered into the search query box, the text area at the bottom left of the ST Browser just above the Search buttons. For example, in Figure 52, the user has pasted some text from one node into the query box and edited it, then clicked the Search button to find the UAF nodes most similar to the text in this query.

Using these techniques enables experienced users who are responsible for maintaining and editing the UAF itself to compare UAF nodes and, perhaps, to edit the content of certain nodes to improve distinctions among similar nodes, to add clarifying examples, etc.

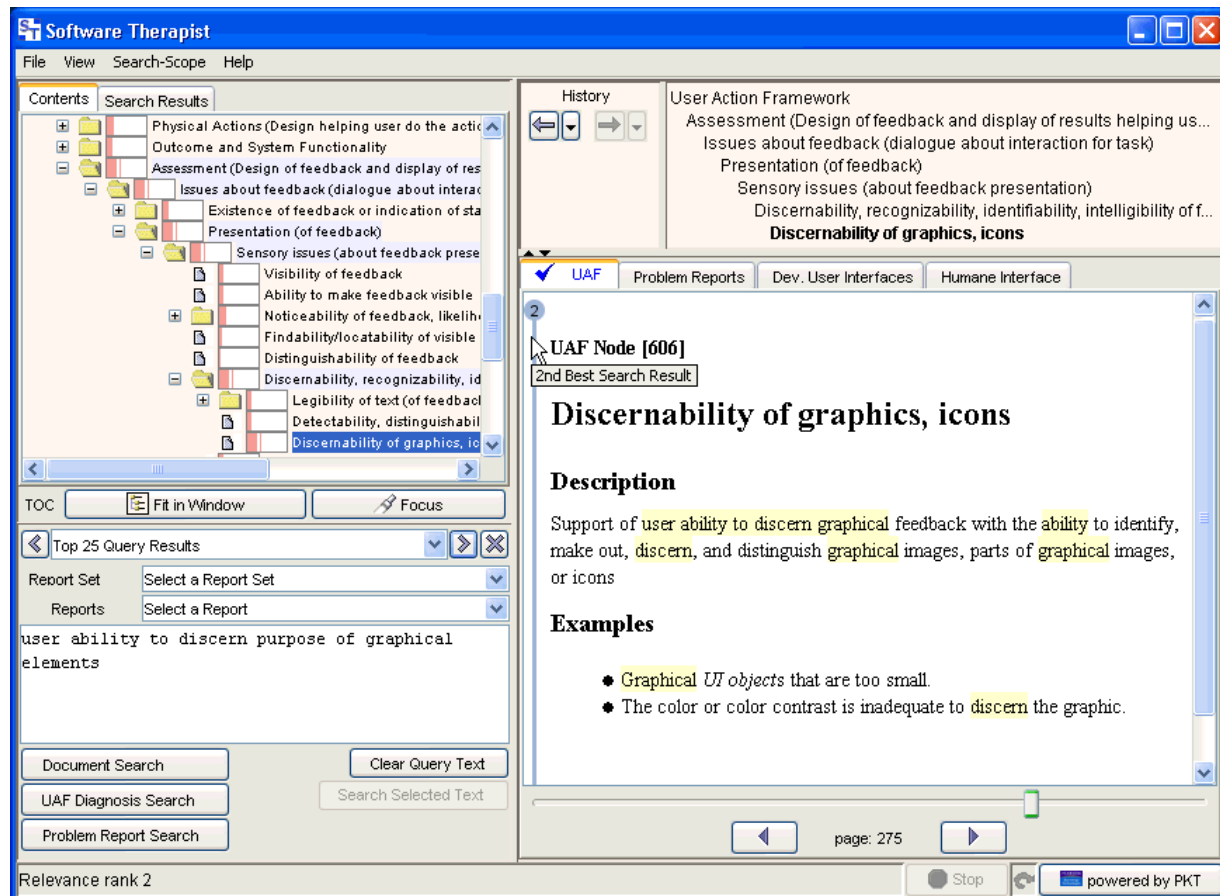


Figure 52. ST Browser showing the 2nd best search result after a “Document Search” for UAF nodes using arbitrary text pasted into and/or edited in the search query box (at bottom left).

3.3 Human Validation of LSA Classification

One of our original goals was to use LSA to search the UAF during problem diagnosis to identify the best candidate diagnosis (or diagnoses). The text of a problem description would be compared to the texts that constitute the UAF, with the best matches indicating the best diagnosis points. In the initial version of this “UAF Diagnosis” function, we used LSA to rank the similarity of each individual node in the UAF to the current problem report. An informal analysis of this approach compared the LSA node rankings for a set of problem reports to the “correct” diagnoses for these reports as established by human experts. The results showed that the top diagnoses by LSA were numerous and not well differentiated from each other and that the best diagnosis according to the human experts was often well down the list produced by LSA.

This result was not entirely unexpected, because many concepts with similar semantics are distributed throughout the UAF. For this reason, many UAF nodes contain highly similar text, using the same terms to describe concepts that are quite similar, but with some important distinction. This distinction may, in fact, not be present in the text of two similar UAF nodes. This is because each node in the UAF constitutes at least a partial diagnosis consisting of the current node and all its ancestor nodes, i.e., the **path** from the root of the UAF to the current node. In this way, two distinct UAF diagnoses may have terminal (or other) nodes that have nearly identical text, yet the diagnoses are distinct because they constitute separate paths through the UAF.

This property of the particular taxonomic organization of the UAF makes it problematic to search for a single, “correct”, node, as many nodes will typically have significantly overlapping semantics. For example, it might be difficult to find a best-node match for the concept of “feathers” in a taxonomical knowledge base about birds, because a large proportion of the nodes throughout the structure might have something to say about feathers. It helps to pursue a node match within a taxonomy structure by traversing it as a tree structure, starting at the root node and proceeding down a level at a time. When a diagnosis or classification choice is made at any given level/depth, it prunes off all the siblings and their corresponding sub-trees. So the match at any given level is made among only the siblings choices at that level, and is conditional, i.e., is made in the context of (as a refinement of) all the choices already made at previous higher levels.

We addressed the problem of high similarity among many UAF nodes in a couple of different ways. First, we implemented a new LSA-based search algorithm in which the text of problem reports (or other text of interest) is compared not simply to UAF nodes, but to possible UAF diagnoses, i.e., possible paths through the UAF. This was implemented as a new search service and new user interface controls in the ST browser. Now, in addition to the ability to search for UAF nodes, users can search for which paths through the UAF—i.e., UAF diagnoses—are most similar to a given problem report. To do this, in a pre-processing step, documents are created for all possible diagnosis paths in the UAF, essentially all possible paths from the root UAF node to every terminal node. In the ST browser, once the user has selected a problem report or entered a problem description in the query field, the “UAF Diagnosis” button may be selected to initiate this type of search. The results are a list of UAF diagnosis paths, ordered by their similarity to the query text. (See, e.g., Figure 29 to Figure 34.)

Although this type of “UAF path” search produced better results than the simple “UAF node” search, it still only rarely produced the correct diagnosis as determined by a human expert as the top result. This means that the usability practitioner must still make a final determination of the best diagnosis, rather than relying on fully automated diagnosis, although this determination may be guided by the results of the UAF path search.

Further complicating the automated diagnosis process was a fact that we discovered in our analysis of large numbers of real-world usability problem descriptions: many of these descriptions lack enough information and precision for even human evaluators to make a proper diagnosis. This led us to additional mechanisms to aid usability engineers in the problem diagnosis process. The first of these was to incorporate support for the techniques of *micro-iteration* and *immediate intention identification* into the DCART component of the Software Therapist system, as described above in sections 2.1.2.3 and 2.1.2.4. These techniques help evaluators get more complete information from user participants while they are still present in the usability testing sessions, which provides crucial information that is required for more complete problem reports and can guide evaluators in deciding among several possible diagnoses for a given problem.

Another enhancement designed to aid the usability practitioner in identifying the best diagnosis for a problem was a further refinement of the UAF path search. Instead of searching against all possible UAF diagnosis paths at once, users may limit the scope of the search to any branch of the large tree structure that constitutes the UAF. This enables users to conduct a hierarchical search for the best path through the UAF, following as much of the human diagnosis process as deemed appropriate by the practitioner, while allowing for suggestive guidance from the LSA-based semantic search of paths through the UAF. So, for example, a user may have already determined which top-level branch of the UAF should be followed for a given problem report (e.g., through immediate intention identification). In this case, the user could limit further diagnosis to this branch of the UAF by selecting its search scope checkbox in the contents panel of the ST browser. (See Figure 35 and Figure 36.)

4 Usability Content Library Collection

4.1 Usability Problem Report Library

In our usability problem library, we currently have 111 sample problems that are fully documented including descriptions, context information, solutions, and UAF diagnoses. Figure 53 shows some of the usability problems in the library.

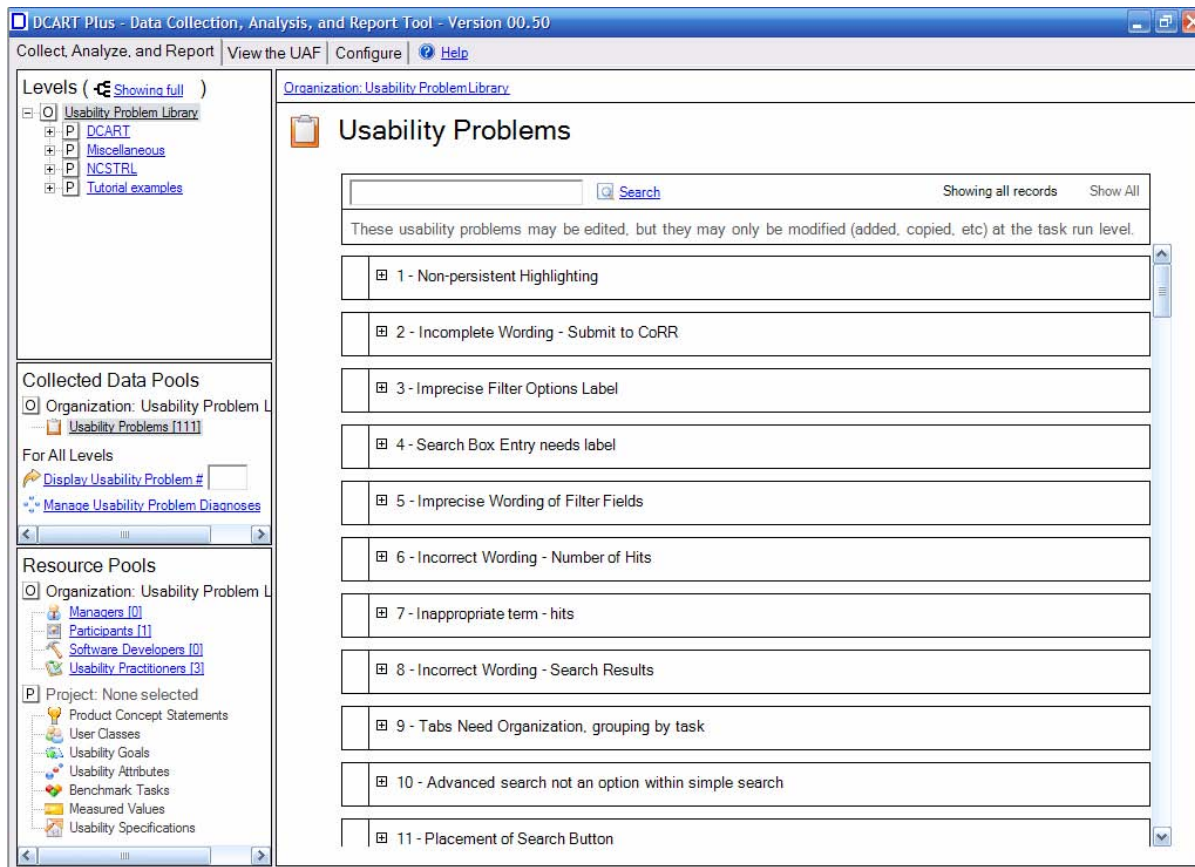


Figure 53. Screenshot of some example usability problems from the UAF usability database.

4.2 UAF Terminal Node Services

UAF terminal node services are the services that are provided to practitioners when they have selected a UAF path to diagnose a usability problem. Usability problem diagnosis is performed on an individual problem to determine problem type, subtype, and causes. The space for usability problems can be described as multidimensional, and diagnosis involves navigating this space to select the dimensions, which best encompass the problem. The primary motivation for diagnosis is that it yields a clear, complete, and

unambiguous statement of the design flaw to be fixed. A secondary motivation for diagnosis is its role in terms of normalizing problem descriptions for comparison and evaluation.

4.2.1 Case Studies

Usability problem diagnosis involves associating a usability problem with the correct usability concept that describes the cause within the interaction design. The UAF is available for browsing and can be used for problem diagnosis from within DCART, as shown in Figure 54, as well as in the ST Browser.

When practitioners use the UAF with DCART to diagnose problems, they have access to two types of case studies. The first is links to sample usability problems in the usability problem report library (Section 4.1). The second is links to usability problems that have already been diagnosed by practitioners working on projects in the same organization.

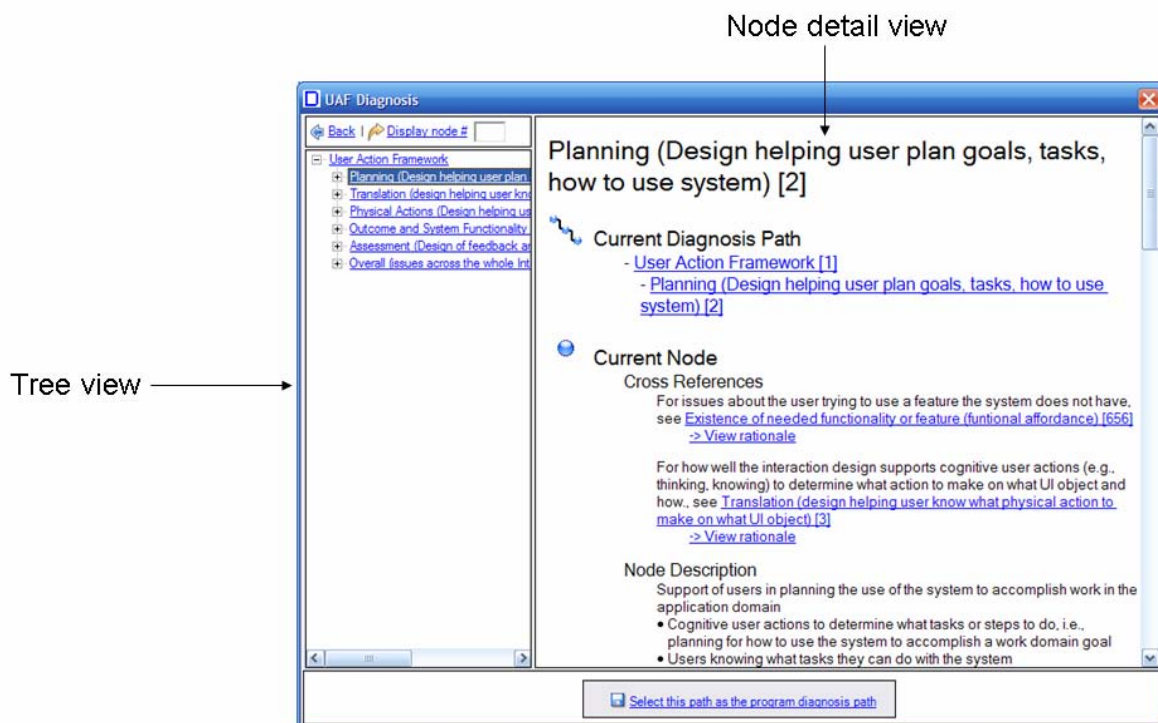


Figure 54: UAF Diagnosis form

There are two ways to traverse the UAF in DCART, each providing the practitioner with different diagnostic information. The first is with the tree view on the left-hand side, which allows practitioners who are familiar with the UAF to quickly traverse the UAF. This method provides the practitioner the main concepts in the UAF and allows them to quickly distinguish among possible diagnoses. Practitioners that are not familiar with the UAF or that want more information about particular concepts can traverse the tree using

the node detail view. This view provides title and path information like the tree view, but it also includes information on other potentially appropriate diagnosis paths and examples of usability concepts.

5 Evaluation

5.1 Exploratory Studies

We conducted a series of exploratory studies to help us understand how problem analysts perform diagnosis with the UAF. The first study focused on the performance of analyst subjects who were new users of the UAF; the second study utilized verbal protocol to help us better understand the diagnosis process by analyst subjects who were intermediate users of the UAF.

The first study was intended to get an indication of how well new users of the UAF could diagnose usability problems with the UAF and what we could do to improve the accuracy of the diagnoses. The study involved 25 graduate usability engineering students who were new users of the UAF. The students used the UAF to diagnose 20 usability problems based on a usability inspection of a kiosk ticket system that they had used in class. The students had two weeks to complete the diagnoses and did it in a time and place of their choosing.

We gave each student a unique username and password pair for the UAF Problem Reporting Tool (PRT), a web-based tool, and told them to use it to report their answers. We entered the 20 usability problems in the PRT as exercise originals. Each exercise original contained a usability problem report and an expert diagnosis path within the UAF. The expert diagnosis path was not visible to students until they had already selected their own diagnosis path and submitted it. Students used the PRT to create their own instances of the exercise originals for submission. When a student created her own instance of an exercise original, she would be presented with a form that contained the usability problem description and an empty text box for the diagnosis. The student would then use the UAF Viewer to find the most appropriate path through the UAF and paste that path into the form. The student could revise each diagnosis as much as she liked until she confirmed it as final. Upon confirmation, the system would present the student with both her diagnosis and the expert's diagnosis from the exercise original. The student then had the opportunity to compare the diagnoses and submit an explanation of the differences.

We compared the students' diagnoses to the expert diagnoses to determine how similar they were. At the time of the study, the top three levels of the UAF were relatively stable, but the lower levels were still being refined. As a result, we considered the students' diagnoses to match those of the experts if they had the same top three levels. We also gave credit for a match if students described how flaws in the usability problem report led to a misdiagnosis.

For some of the usability problem reports, students consistently selected the wrong top-level node. As we read through the students' rationales for selecting a different top-level node, it became clear that the usability problem report did not provide the necessary information for them to distinguish reliably between stages in the Interaction Cycle. For example, a particular usability problem report read, "The color coding scheme for seats is problematic for individuals with red/blue color blindness. In addition, in the detailed seat view, purple isn't noticeable as a color." The expert diagnosis had the Translation stage as its top-level node because the expert considered the poor color coding to affect the user's ability to determine what to do next in the task of selecting a seat for a theatre performance. The expert believed that color blindness would prevent users from recognizing that the seats were in fact selectable objects. The students did not have this information because it was not explicitly recorded in the usability problem report, and they assumed that it was a Physical Actions usability problem that resulted from the inability of a color blind user to determine the availability of a seat based on the colors of red, blue, and purple. For

example, one student's rational read, "I chose this [Physical Actions] because I assumed that the person knew how to select the seats".

Examples such as this helped us to realize that the usability problem reports did not contain all the information necessary to correctly diagnose a usability problem. We decided to run another study to determine if more experienced UAF users would have the same difficulties. This second study used verbal protocols taken from six usability engineering graduate students. These students were intermediate users of the UAF who had participated in a training session.

We worked with each student for two hours. The students used the PRT to diagnose usability problems from professional usability engineering labs in the same manner as in the first part of the study. Via verbal protocol, we asked the students to talk us through the node decision process and tell us when they felt that they were confused. As in the first study, the students had trouble deciding between stages of the Interaction Cycle for some of the usability problem reports.

The inability of the students to choose the correct top-level category was directly related to the lack of necessary information in the usability problem reports. This lack of information is particularly problematic given the fact that the usability data collection stage and the analysis stage, which includes diagnosis, are separate in typical usability evaluation sub-processes.

5.2 Analogy to Medical Diagnosis

The need for problem diagnosis is not new with usability engineering; it is central to any domain that involves finding and fixing problems, including automobile repair and the medical field. In medicine, a nurse might see the patient, gather some initial data via common measurements such as temperature and blood pressure, and take a statement of the patient's complaint. The doctor will review this initial information, possibly making more measurements and observations, and will probably ask the patient to repeat a description of the complaint. Throughout the case, the doctor draws on a structured knowledge base of medical concepts and issues that relates symptoms with diseases and serves as a guide to formulating potential diagnoses (diagnostic hypotheses).

Even while the patient is still in the examining room, the doctor begins to use the medical knowledge framework to highlight common and distinguishing characteristics among the potential diagnoses and to determine and ask questions that represent additional information necessary to rule in or rule out each of these diagnostic hypotheses.

This initial analysis then drives further data collection as the doctor makes more measurements and observations (e.g., looks in the patient's throat) and asks the patient more questions (e.g., about symptoms, background), seeking to prune the hypotheses. This "micro-iteration" (using our term) of data collection with analysis taps information that was not collected initially but is still available (for example, by asking the patient or, if necessary, bringing the patient back for a return visit), just when it is needed for diagnosis. The medical procedure supports micro-iteration but, while the typical usability engineering cycle is iterative overall, it does not support micro-iteration between data collection and analysis.

Our concept of micro-iteration has a counterpart in a type of reasoning used by medical doctors called hypothetico-deductive reasoning, which involves generating several diagnostic hypotheses and then working backwards to prove or disprove them. While working backwards, it may be necessary to collect additional data to disambiguate or distinguish a hypothesis. Doctors interact with the patients to develop hypotheses about potential diseases, identifying key distinguishers to determine what tests are needed to get critical data to rule in or rule out these diseases. The

diagnosis process is iterative and doctors continue to interact with patients until they get the necessary data.

5.3 Formative Studies of the Wizard

The exploratory studies and comparisons to medical diagnosis helped us determine the need for micro-iteration. We use the term *immediate intention*¹² to refer to the necessary information that usability practitioners need to identify and record during micro-iteration. More specifically, immediate intention provides information about what the participant is doing when he or she encounters a design flaw that results in a usability problem.

Evaluators need some kind of support in asking the right questions to elicit immediate intention information. The UAF has proved to be useful in structuring the process of capturing missing usability problem data, but the UAF is intended for use in the analysis stage and is probably too bulky and time-consuming for use by most evaluators for initial diagnosis as part of the usability data collection stage. As a result, we developed the Wizard, a lighter-weight tool that is limited to the top-levels of the UAF (the Interaction Cycle) and tailored specifically for helping evaluators to quickly identify the immediate intention associated with a usability problem during micro-iteration.

We performed two formative studies of the Wizard. Because these studies were formative with the primary goal of improving the Wizard design, we report here our lessons learned only as informal observations or intuitive insights, and not as formal results. Our observations during these studies suggested to us that the Wizard has potential as an effective tool for helping evaluators determine the correct stage of the Interaction Cycle for a given usability problem.

For the first formative study, we developed a static prototype (a series of linked web pages) of the Wizard that had abstract descriptions of stages of the Interaction Cycle and concrete examples. Table 1 contains the pairs of questions presented at each decision point. The participants for the first study, who all had some general usability knowledge, included an individual who had never used the UAF, two beginning users, one intermediate user, and two experts. The participants were given 10 usability problem reports with varying levels of immediate intention specified. After the participants read a usability problem description, we first asked them to choose a stage in the Interaction Cycle and then had them use the Wizard. The participants had one hour to complete as many identifications as they could and were allowed to skip usability problem descriptions and return if they had time.

Table 1. First version of the Wizard.

<p>Is your problem one that is internal to the system and invisible to users?</p> <p>For example, does the system automate too much and take control away from the user?</p> <p>(Outcome and System Functionality)</p>	<p>Does your problem concern the user's interaction with the user interface?</p> <p>For example, is your problem related to the user's ability to plan for his task, determine appropriate interface elements for that task, manipulate those interface elements, or make sense of the results his actions?</p>
--	---

¹² See section 2.1.2.3.

<p>Is your problem independent of the Interaction Cycle?</p> <p>For example, does the problem deal with interaction flaws that occur throughout the system?</p> <p>(Overall)</p>	<p>Does your problem deal with a specific stage in the Interaction Cycle?</p> <p>For example, does your problem deal with an interaction flaw that occurs in one place?</p>
--	---

<p>Is your problem about actually performing physical actions on interface objects?</p> <p>For example, does the user have problems manipulating interface objects?</p> <p>(Physical Actions)</p>	<p>Is your problem about cognition or the user's ability to understand how to use the system?</p> <p>For example, does the user have trouble determining what interface objects mean?</p>
---	---

<p>Is your problem concerned with the user's understanding after he made an action?</p> <p>For example, does the user have trouble understanding feedback from the system?</p> <p>(Assessment)</p>	<p>Is your problem concerned with the user's understanding before he makes an action?</p> <p>For example, does the user have trouble determining how to perform a task?</p>
--	---

<p>Is your problem about how well the system supports the user in planning use of the system to accomplish a task?</p> <p>For example, can the user determine what they can do with the system?</p> <p>(Planning)</p>	<p>Does your problem concern the user's ability to determine (know or not know) how to do a task step?</p> <p>For example, does the user know what physical actions to make on which user interface objects?</p> <p>(Translation)</p>
---	---

The results of the first study are shown in Table 2. Participants P1, P3, and P5 did not complete all the identifications. A forward slash indicates the number of correct identifications as compared to the total number attempted. The non-UAF user did not feel that he or she was capable of selecting a stage in the Interaction Cycle first and only used the Wizard. Table 2 also contains counts of the number of correct identifications confirmed by the Wizard, the number of incorrect identifications corrected by the Wizard, and the number of times the participant was led astray by the Wizard after making a correct diagnosis (Confirmed, Corrected, and Led Astray, respectively).

Table 2: Results of the first Wizard study

	P1 beg	P2 beg	P3 non	P4 ex	P5 in	P6 ex
W/O Wizard - Correct	3/5	6/10		10/10	7/8	10/10
With Wizard - Correct	1/5	6/10	4/7	10/10	7/8	10/10
- Confirmed	1	4		10	7	10
- Corrected	0	2		0	0	0
- Led astray	2	2		0	0	0

non = non-UAF, beg = beginner, in = intermediate, ex = expert

The participants who were expert users (P4 and P6) of the UAF identified the correct stage of the Interaction Cycle for all usability problems and then confirmed their choices with the Wizard. After using the Wizard a few times, they began to focus only on key words in the abstractions and used the Wizard much more rapidly. The intermediate user (P5) spent more time describing the decision process than did the expert users (as part of verbal protocol) and did not complete all 10 usability problems. On the completed usability problems, however, the intermediate user performed almost as well as the expert users. These results suggest that the Wizard helps the advanced users of the UAF improve their identification speed and associate words and concepts with stages of the Interaction Cycle.

One beginning user (P1) performed particularly poorly with the Wizard, while the other (P2) performed well overall. They both, however, were led astray twice by the wording in the Wizard, which indicated that it required improvement. The non-UAF user (P3), who had no experience with the UAF and no training, was able to use the Wizard to make four correct identifications. This result suggests that the Wizard could be a useful training tool.

The feedback provided by the participants led us to develop a second version of the Wizard. (See Table 3.) In this version, the most important change was in the wording of the questions and examples. Verbal protocol in the first study revealed that certain words and phrases confused the participants and lead to misdiagnoses. For example, the question for the Outcome and System Functionality stage in the first version read: “Is your problem one that is internal to the system and invisible to the user?” The participants, particularly those with limited experience with the UAF, did not understand the phrase “internal to the system.” Several times, in fact, these participants selected this choice when the usability problem was not related to functional issues, because they thought that “internal” meant any processing by the system. Because most interactions involve processing by the system, they made mistakes. We corrected the problem in the second version by specifying that the Outcome category referred to backend functional issues not in the user interface software.

Table 3: Second version of the Wizard

<p>Is your problem in the non-user interface software (e.g., a bug in the back end computation)?</p> <p>For example, does the system automate too much and take control away from the user?</p> <p>(Outcome and System Functionality)</p>	<p>Does your problem concern the user's interaction with the user interface?</p> <p>For example, is your problem related to user planning, determining actions, making actions, or understanding feedback?</p>
<p>Does your problem cut across the whole Interaction Cycle and not just a particular part?</p> <p>For example, does the problem deal with interaction flaws that occur in several places in the user interface?</p> <p>(Overall)</p>	<p>Does your problem deal with a specific stage in the Interaction Cycle?</p> <p>For example, is your problem related to user planning, determining actions, making actions, or understanding feedback?</p>
<p>Is your problem about actually performing physical actions on interface objects or with devices?</p> <p>For example, does the user have problems finding or seeing an object to click or actually performing the clicking and dragging?</p> <p>(Physical Actions)</p>	<p>Is your problem about cognition (thinking, knowing) or the user's ability to understand how to use the system?</p> <p>For example, is your problem related to user planning, determining actions, or understanding feedback?</p>
<p>Is your problem concerned with the user's ability to understand the outcome of an action after he made the action?</p> <p>For example, does the user have trouble understanding feedback from the system?</p> <p>(Assessment)</p>	<p>Is your problem concerned with the user's understanding of what action to take and/or how to do an action before he makes the action or the next appropriate action?</p> <p>For example, does the user have trouble determining how to perform a task or the next appropriate task?</p>
<p>Is your problem about how well the system supports the user in high-level planning use of the system to accomplish a task?</p> <p>For example, can the user make an overall</p>	<p>Does your problem concern the user's ability to determine (know or not know) how to do a specific task step?</p> <p>For example, does the user know what</p>

general plan for using the system? (Planning)	physical action to make on which user interface object? (Translation)
--	--

For the second Wizard study, we randomly selected five participants who had some familiarity with usability engineering, but who were not familiar with the UAF. The first study strengthened our belief that intermediate and expert UAF users can do well identifying the correct node of the UAF to specify immediate intention with the Wizard, and we wanted to see if rank novices (with respect to the UAF and the Wizard) could do the same. For this study, we gave each participant a five minute training course on the Interaction Cycle, so that they could select a stage without using the Wizard to avoid the situation that we had with P3 in the first study. Each new participant was given the same 10 usability problem reports that we used in the first study. For the first five usability problem reports, we had the participants first choose a stage of the Interaction Cycle that specified the immediate intention in the usability problem description without the Wizard and then with the Wizard. This allowed us to test the Wizard's ability to provide confirmation for correct identifications and help users after incorrect identifications. For the next three usability problem reports, we had the participants use only the Wizard, which allowed us to evaluate the Wizard's ability to help users select the correct stage of the Interaction Cycle the first time. For the last two usability problem reports, we had the participants select a stage in the Interaction Cycle without the Wizard and then tell us words and phrases that they had learned while using the Wizard that had helped them make a decision. With these usability problems we hoped to indirectly evaluate what the participants had learned.

Table 4 shows the results of the second Wizard study. The results suggest that the second version of the Wizard helped new users of the UAF identify the correct stage of the Interaction Cycle. Only one participant (P3) was led astray once by the Wizard. In addition, all participants correctly identified the last two usability problem descriptions without the Wizard, which suggests that they had learned from the Wizard and were incorporating the concepts that help to distinguish between stages of the Interaction Cycle.

Table 4: Results of the second Wizard study

	P1	P2	P3	P4	P5
Problems 1-5					
- W/O Wizard	2	1	3	3	3
- With Wizard	3	3	2	4	3
-- Confirmed	2	1	2	3	3
-- Corrected	1	2	0	1	0
-- Led astray	0	0	1	0	0
Problems 6-8					
- Correct	2	3	3	2	2
Problems 9-10					
- Correct	2	2	2	2	2

5.4 Formative Evaluation

5.4.1 Low-Fidelity DCART Prototype

We began prototyping DCART as a low-fidelity web application. We iterated between design and formative evaluation activities to gradually improve the functional capabilities and interaction design. The evaluation sessions consisted of design walkthroughs by the evaluators and informal sessions with students at Virginia Tech. We produced a sequence of seven low-fidelity versions.

We formally evaluated the seventh version with 11 participants. The participants performed 13 benchmark tasks with DCART. (See Appendix XXX, section 11). We recorded 20 usability problems. The main issues experienced by participants were related to the organization of resources. We had not yet developed the idea of hierarchical context and resources were collected inside of levels, which effectively hid them from the participants. In addition, we had just began developing the concept of the expanding list view for records, and participants had great difficulties expanding and editing records.

5.4.2 Formative Evaluation of the ST Browser

A formative evaluation of the ST Browser was conducted by researchers at Virginia Tech. In this evaluation, two expert users of the UAF used the ST Browser to perform problem diagnoses of approximately 20 problem reports and recorded their observations of apparent functional bugs, usability issues, and the performance of the LSA-based UAF diagnosis search functionality. They concluded that the browser provided powerful and easy-to-use navigation of the UAF, that the combination of views (TOC, search and diagnosis, path summary, and primary content) was effective, but that the search features were disappointing in their failure to identify the correct diagnosis path through the UAF in most cases.

The evaluators concluded that this shortcoming is due in large part due to the fact that initial problem descriptions are typically vague and incomplete.[REF?] That is, they do not contain enough information by themselves to lead to a correct diagnosis, even for a human expert. Part of the process of usability problem diagnosis is to add enough information to an initial description, using a common vocabulary, to fully identify what kind of problem is involved, what flaw in the design caused the problem, and how it may best be fixed. In the context of the UAF, the problem diagnosis process leads the analyst to think of the most important problem dimensions and to describe the most relevant attributes in those dimensions; but rarely can the initial problem description by itself lead to this information.

It is this observation—that initial problem descriptions alone are inadequate to determine a full and correct UAF problem diagnosis—that led us in two new directions: (1) the modifications of the usability data management cycle to help analysts do critical additional data collection early in the process (section 2.1.2 above), and (2) modifications of support tools and their use to better support determination of the best analysis path through the UAF. One of these tool modifications was the addition of the Diagnosis Wizard (section 2.1.2.5).

Another modification is that we no longer expect users of the ST Browser to be able to rely on a *fully* automated diagnosis to be correct. That is, simply taking the top result from a UAF

diagnosis search in the ST Browser is unlikely to be sufficient, because there are so many possible paths through the UAF, and we have found in practice that, for most initial problem reports, there are many paths returned from the search that are not well differentiated from one another: many paths have about the same degree of semantic similarity to the query (initial problem report); the best diagnosis path doesn't stick out from the crowd of similar paths. We attribute this to (1) the inadequacy of typical problem reports, as described above, and (2) the complexity in structure and content of the UAF.

There are several ways to address this problem, some of which we have implemented in the Software Therapist system and others which could be pursued in future work. With regard to the inadequacy of initial problem reports, we have already described procedural changes in initial data collection, supported by DCART, that may help improve the content of initial problem reports and help the analyst to make a more accurate determination of a user's immediate intention and, thereby, make a more accurate determination of the first segment of the UAF diagnosis path.

However, even with these changes, it still appears unrealistic to expect analysts in most circumstances to provide an initial problem description that is complete and detailed enough to adequately support fully automated diagnosis with the UAF. Instead, the analyst will need to contribute to and make the final decision regarding what is the best diagnosis, perhaps enhancing the problem description along the way. Although this can still be done on an entirely manual basis—at least by experts, we have also designed the search capabilities and user interface of the ST Browser to support this sort of hybrid approach, in which use LSA-based search may be used multiple times and in different ways to assist the analyst in arriving at a proper diagnosis. These design features include multiple types of LSA-based search (for similar UAF nodes, UAF paths, similar passages in related literature, and semantically related problem reports) and the ability to limit the scope of any of these search types, though in particular for the UAF diagnosis path search. This allows the analyst, for example, to engage the assistance of LSA only below the initial level (since that may have already been decided) or only on certain sub-branches at any point in the UAF. This lends itself to an exploratory approach in which the analyst may examine multiple paths (and partial paths) through the UAF, related literature, and similar problem reports, all with the assistance of LSA-based semantic search.

5.5 Field Trials

5.5.1 Field Test of High-Fidelity Prototype Version

After development and testing of the low-fidelity version of DCART, we decided to begin developing a high-fidelity prototype in C#. As with the low-fidelity prototype, we iterated between design and evaluation activities to gradually improve the functional capabilities and interaction design. We evaluated the first four versions locally with walkthroughs and informal sessions with students.

We conducted field tests of the fifth version with cadets at the USAFA in the fall of 2005. The cadets used the tool as part of a class assignment and recorded critical incidents as they used the tool. The feedback helped us identify several key problems with the tool. For example, the cadets had difficulty understanding where and how DCART saves data. DCART uses a database to store data and performs incremental saves as individual records are edited. An overall save operation is not necessary. We need to provide cognitive affordances to explain this data model to DCART users.

We updated the high-fidelity prototype to address the problems identified by the USAFA cadets and then performed a round of field testing with usability practitioners using this version in the winter of 2006. The exploratory tasks for participants are available in Appendix XXX (section 12). The usability practitioners were very helpful and provided us with a large amount of feedback. We summarize this feedback in the following paragraphs.

5.5.1.1 Typical run-of-the-mill usability problems

As expected, there were reports on numerous of the usual type of usability problem reports about details. These were not really what we were looking for in this field test. For example:

- Need a better cognitive affordance to distinguish the difference between a project and a version.
- Need qualifying words added to labels to be more precise about meaning of fields, such as Start Date and End date (start and end of what?).

5.5.1.2 Positive feedback

They liked the fact that DCART kept track of usability problems and most liked the usability problem collection form, but felt that this functionality should be made the heart of the tool. They also liked the fact that the tool integrated usability problem analysis and wasn't just an aid for keeping lists of usability problems.

5.5.1.3 Constraints

We went to considerable lengths to build in constraints to help users avoid deleting important data or relationships among resources. However, we learned that it was not a good idea to try to second guess the way users would want to use the system. The constraints were essentially hidden from the users and only served to confuse them when they couldn't do something but didn't know why. They felt that the data relationships were too rigid (e.g., inability to enter a problem record unless it is associated with a Task Run). In many cases, the constraints back-fired and posed a barrier to users who wanted to work “outside the box” (at least outside the box we envisioned). At the risk of a user occasionally doing something destructive, we have to ease the constraints and let users do whatever they want to, in whatever order they want to.

5.5.1.4 Complexity

The large and complex structure built into the data was overwhelming to users and, in some cases, did not match their needs or their model of the structure. Most participants said that the model of the tool as seen through our user interface was just too complex. As one example, they had trouble associating usability problem data with Task Runs. This could be helped in two ways:

- Less emphasis on a data structure orientation
- Some kind of data structure “map”, to show what parts are connected and how and what parts are “dangling” and possibly need to be connected eventually

5.5.1.5 Work flow model rather than data structure model

Users wanted to see their own work flow better reflected in the tool. They wanted to see their working environment wrapped around them when they are working within the tool. Instead, they felt they saw too much structure, too much focus on the data and not enough on their process. The data was broken up into all kinds of different data objects and the data connections were never visible or obvious enough. They wanted to see the tool organized around their work activities,

especially usability data collection and analysis, and not all the up-front setup material that often seemed to get in the way.

One participant wanted the work flow to reflect the user's time line. This meant an ability to see versions, sessions, task runs, and usability problem occurrences organized on a line of time that matches how they occurred within a project and (for the problems) how the user encountered them. She wanted to map all of her data collection and analysis activities to the user's time line, much as they do in Morae¹³.

In a word, tool users wanted more flexibility in being able to do whatever they want to do at a given moment and maybe connect that back to the overall project structure later, if ever. One possible way to do this is to include more of a wizard look to the user interface, asking questions about what the DCART user wants to do and giving them more choices about doing that, in or out of the context we were trying to provide.

5.5.1.6 Expert vs. Novice Users

Most of these participants complained of the process reflected in the tool as being too heavy and effort-intensive, taking away from the essential business of collecting data and making it unsuitable for use in everyday usability testing environments. The question might be how to streamline it for expert users without losing what we still perceive as value for novice users.

The people we chose as participants in this trial are top usability experts, not typical of the average or novice practitioner. Yet, we made a conscious effort in DCART to target novice usability practitioners, in an effort to help them perform more like experts. Our primary goal was effectiveness and quality of output, while efficiency was secondary. To our experts, efficiency was essential.

Our expert participants could provide high quality problem reports without a tool reminding them of this and that and guiding them through designer knowledge requirements, immediate intention, and diagnosis. So, it is natural for them to complain about the extra overhead in a process they already do so well. For them we need an expert mode, hiding everything except the essentials and getting out of the way of their work flow.

However, we are still left wondering about the novice practitioners. If saving time for them means low quality usability reports, then we are back to the situation that motivated us to do this work in the first place. Thus, we are led to believe we need modes or other design approach that will serve both kinds of users. For novices, perhaps the tool needs more wizard-like hand-holding. For experts, perhaps it needs fewer constraints and to be less directly tied to the hierarchical structure.

We feel that this is a normal kind of path in the development of a complex tool such as DCART and that we could not have gotten this kind of guidance from practitioners in the field without going through an iteration of a concrete high-fidelity prototype for them to try and react to, in the context of how they work in the field. Perhaps the present version may be more suited to education than professional practice. Most participants felt that the general concepts embodied in the tool were valuable and the very weight of the structure and detail that made it impractical for practice in the field might make it ideal as a teaching aid, to convey the concepts in a usability engineering course.

¹³ See <http://www.techsmith.com/morae.asp>.

6 Conclusion

In the work reported here, we have addressed the problem of low return on usability investment and offer a means of support for usability practitioners in identifying, understanding, documenting, and fixing usability problems. We have done this by (1) validating and extending a structured knowledge framework of usability concepts for organizing and relating usability data to design flaws and solutions (the UAF), (2) specifying a usability data management cycle to support a diagnosis process that is iteratively interleaved with data collection, and (3) developing a software system for usability engineering practitioners that includes components to support the activities of usability data collection, organization, and problem analysis and reporting, as well as automated support for usability problem diagnosis using a sophisticated statistical technique for the analysis of text. All three of these tasks were accomplished by combining the unique strengths of the User Action Framework (UAF) and Latent Semantic Analysis (LSA).

The software system developed is called the Software Therapist and consists of two major components: the Data Collection, Analysis, and Reporting Tool (DCART) and the Software Therapist (ST) Browser. The system constitutes a usability engineering problem solving environment based on the UAF and LSA that assists usability practitioners in setting up and conducting usability trials and walkthroughs, and in collecting, storing, and analyzing usability data. In particular, it supports practitioners in the process of usability problem analysis, i.e., the process of accurately diagnosing a particular usability problem, and analyzing that problem in the context of the UAF, other problem reports, and a collection of related online literature in order to determine an appropriate solution to the problem. This process is supported by several features, including the Diagnosis Wizard and specialized search capabilities based on LSA. The LSA features are made available in the ST Browser, and provide “deep” semantic search functions to find material related to usability problem reports or any other textual query, including related nodes in the UAF, paths constituting diagnoses within the UAF, and semantically related passages in the problem report database and in available usability/HCI literature resources. Several versions of these tools were tested during development in usability labs, with students at the US Air Force Academy, and with usability professionals.

7 References

- ACM. (2000). Communities Centered Around Knowledge. In Proceedings of the ACM Hypertext 2000 Conference (the Eleventh ACM Conference on Hypertext and Hypermedia), ACM: New York.
- Andre, T., Hartson, H. R., Belz, S., & McCreary, F. (2001). The user action framework: A reliable foundation for usability engineering support tools. *International Journal of Human-Computer Studies*, 54 (1), 107–136.
- Andre, T. S., Hartson, H. R., & Williges, R. C. (2001b). Determining the effectiveness of the usability problem inspector: A theory-based model and tool for finding usability problems. *Human Factors* Vol. 45.
- Andre, T. S., Belz, S. M., McCreary, F. A., & Hartson, H. R. (2000). Testing a Framework for Reliable Classification of Usability Problems. In Proceedings of the Human Factors and Ergonomics Society 44th Annual Meeting, Human Factors and Ergonomics Society: San Francisco, CA, 573–577.
- Andre, T. S., Hartson, H. R., & Williges, R. C. (1999). Expert-based usability inspections: Developing a foundational framework and method. In Proceedings of the the 2nd Annual Student's Symposium on Human Factors and Ergonomics of Complex Systems, North Carolina A&T State University: Greensboro, NC, 142–148.
- Andre, T. S., Williges, R. C., & Hartson, H. R. (1999b). Effectiveness of usability evaluation methods: Determining the appropriate criteria. In Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting, Human Factors and Ergonomics Society: Santa Monica, CA, 1090–1094.
- Berry, M. W. (1992). Large scale singular value computations. *International Journal of Supercomputer Applications*, 6(1), 13–49.
- Berry, M. W., Dumais, S. T. and O'Brien, G. W. (1995) Using linear algebra for intelligent information retrieval. *SIAM: Review*, 37(4), 573–595.
- Bødker, S. (1989). A human activity approach to user interfaces. *Human-Computer Interaction*, 4, 171–195.
- Bødker, S. (1991). Through the interface: A human activity approach to user interface design. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Brooks, P. (1994). Adding value to usability testing. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 255–271). New York: John Wiley & Sons.
- Butler, K. A. (1996). Usability engineering turns 10. *interactions*, 3 (1), 58–75.
- Coccaro, N. and Jurafsky, D. (1998). *Proceeding of the International Conference on Spoken Language Processing (ISSLP-98)*, 6, 2403–2406.
- Cockton, G., & Lavery, D. (1999). A framework for usability problem extraction. In Proceedings of the Proceedings of the IFIP Seventh International Conference on Human-Computer Interaction – INTERACT '99, IOS Press: Edinburgh, Scotland, 344–352.

- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20 (1), 37–46.
- Cuomo, D. L. (1993). A methodology and encoding scheme for evaluating the usability of graphical, direct manipulation style interfaces. In *Proceedings of the Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting*, Human Factors and Ergonomics Society: Seattle, WA, 1137–1141.
- Cuomo, D. L. (1994). A method for assessing the usability of graphical, direct-manipulation style interfaces. *International Journal of Human-Computer Interaction*, 6 (3), 275–297.
- Cuomo, D. L., & Bowen, C. D. (1992). Stages of user activity model as a basis for user-system interface evaluations. In *Proceedings of the Proceedings of the Human Factors Society 36th Annual Meeting*, Human Factors and Ergonomics Society: Atlanta, GA, 1254–1258.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing By Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6), 391–407.
- Desurvire, H. W. (1994). Faster, Cheaper! Are usability inspection methods as effective as empirical testing? In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 173–202). New York: John Wiley & Sons.
- Dumais, S. T. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23(2), 229–236.
- Dumais, S. T. (1994). Latent semantic indexing (LSI) and TREC-2. In D. Harman (Ed.), *National Institute of Standards and Technology Text Retrieval Conference*. NIST special publication.
- Dutt, A., Johnson, H., & Johnson, P. (1994). Evaluating evaluation methods. In G. Cockton & S. W. Draper & G. R. S. Weir (Eds.), *People and computers IX* (pp. 109–121). Cambridge, UK: Cambridge University Press.
- Fitts, P. (1954). Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 47 (6), 381–391.
- Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76 (5), 378–382.
- Foltz, P. W., Kintsch, W. & Landauer, T. K. (1998). The measurement of textual coherence with Latent Semantic Analysis. *Discourse Processes*. 25, 285–308.
- Gray, W. D., & Salzman, M. C. (1998). Damaged merchandise? A review of experiments that compare usability evaluation methods. *Human-Computer Interaction*, 13 (3), 203–261.
- Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM*, 32, 1164–1173.
- Hartson, H. R. (2003). "Cognitive, Physical, Sensory, and Functional Affordances in Interaction Design." *Behaviour and Information Technology* 22(5): 315-338.
- Hartson, H. R., T. S. Andre, et al. (2003). "Criteria for evaluating usability evaluation methods." *International Journal of Human-Computer Interaction* 15(1): 145-181.

- Hartson, H. R., Andre, T. S., Williges, R. C., & van Rens, L. (1999). The User Action Framework: A theory-based foundation for inspection and classification of usability problems. In H. Bullinger & J. Ziegler (Eds.), *Human-computer interaction: Ergonomics and user interfaces (Proceedings of the 8th International Conference on Human-Computer Interaction, HCI International '99)* (Vol. 1, pp. 1058–1062). Mahway, NJ: Lawrence Erlbaum Associates.
- Hartson, H. R., & Hix, D. (1989). Human-computer interface development: Concepts and systems for its management. *ACM Computing Surveys*, 21 (1), 5–92.
- Hix, D., & Hartson, H. R. (1993). *Developing user interfaces: Ensuring usability through product & process*. New York: John Wiley & Sons, Inc.
- Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of the CHI '91 Conference Proceedings*, ACM Press: New Orleans, LA, 119–124.
- John, B. E., & Marks, S. J. (1997). Tracking the effectiveness of usability evaluation methods. *Behaviour and Information Technology*, 16, 188–202.
- Karat, C. (1994). A comparison of user interface evaluation methods. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 203–233). New York: John Wiley & Sons.
- Kaur, K., Maiden, N., & Sutcliffe, A. (1999). Interacting with virtual environments: An evaluation of a model of interaction. *Interacting with Computers*, 11, 403–426.
- Kaur, K., Sutcliffe, A. G., & Maiden, N. A. M. (1999). Towards a better understanding of usability problems with virtual environments. In *Proceedings of the INTERACT 99, IFIP/IOS Press: Amsterdam*, 527–535.
- Keenan, S. L. (1996). Product usability and process improvement based on usability problem classification. Unpublished Ph.D. dissertation, Virginia Tech, Blacksburg, VA.
- Keenan, S. L., Hartson, H. R., Kafura, D. G., & Schulman, R. S. (1999). The Usability problem taxonomy: A framework for classification and analysis. *Empirical Software Engineering*, 4 (1), 71–104.
- Laham, D., Bennett, W. and Landauer, T. K. (2000). “An LSA-based software tool for matching jobs, people, and instruction.” *Interactive Learning Environments* 8(3): 171–186.
- Landauer, T. K. (1998). Learning and representing verbal meaning: The Latent Semantic Analysis theory. *Current Directions in Psychological Science*. 7: 161–164.
- Landauer, T. K. (1999). Latent Semantic Analysis: A theory of language and mind. *Discourse Processes* 27. 303–310.
- Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, 104, 211–240.
- Landauer, T. K., Foltz, P. & Laham, D. (1998). An introduction to Latent Semantic Analysis, *Discourse Processes*. 25, 259–284.
- Landauer, T. K., Laham, D., Derr, M. (2004). “From paragraph to graph: Latent semantic analysis for information visualization.” *Proceedings of the National Academy of Sciences*, vol. 101, suppl. 1, 5214–5219.

- Lewis, C., Polson, P., Wharton, C., & Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of the CHI '90 Conference Proceedings*, ACM Press: Seattle, WA, 235–242.
- Lim, K. H., Benbasat, I., & Todd, P. (1996). An experimental investigation of the interactive effects of interface style, instructions, and task familiarity on user performance. *ACM Transactions on Computer-Human Interaction*, 3 (1), 1–37.
- Logan, G. D. (1988). Toward an instance theory of automatization. *Psychological Review*, 95 (4), 492–527.
- Mack, R., & Montaniz, F. (1994). Observing, predicting, and analyzing usability problems. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 295–339). New York: John Wiley & Sons.
- MacKenzie, S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7, 91–139.
- Mayhew, D. J. (1992). *Principles and guidelines in software user interface design*. Englewood Cliffs, NJ: Prentice-Hall.
- Muller, M. J., Dayton, T., & Root, R. (1993). Comparing studies that compare usability assessment methods: An unsuccessful search for stable criteria. In *Proceedings of the INTERCHI '93 Conference Proceedings (Adjunct)*, ACM Press: Amsterdam, 185–186.
- Nayak, N. P., Mrazek, D., & Smith, D. R. (1995). Analyzing and communicating usability data: Now that you have the data what do you do? *ACM SIGCHI Bulletin*, 22–30.
- Nielsen, J. (1989). Usability engineering at a discount. In G. Salvendy & M. J. Smith (Eds.), *Design and using human computer interfaces and knowledge based systems* (pp. 394–401). Amsterdam: Elsevier Science.
- Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In *Proceedings of the CHI '92 Conference Proceedings*, ACM Press: Monterey, CA, 373–380.
- Nielsen, J. (1993). *Usability engineering*. Boston: Academic Press.
- Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 25–62). New York: John Wiley & Sons.
- Nielsen, J., & Mack, R. L. (Eds.). (1994). *Usability Inspection Methods*. New York: John Wiley & Sons.
- Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the CHI '90 Conference Proceedings*, ACM Press: Seattle, WA, 249–256.
- Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 31–61). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Pernice, K., & Butler, M. B. (1995). Database support for usability testing. *interactions*, 2 (1), 27–31.
- Rubin, J. (1994). *Handbook of usability testing: how to plan, design, and conduct effective tests*. New York: Wiley.
- Sears, A. (1997). Heuristic walkthroughs: Finding the problems without the noise. *International Journal of Human-Computer Interaction*, 9 (3), 213–234.

- Shneiderman, B. (1998). *Designing the user interface: Strategies for effective human-computer interaction* (3rd ed.). Reading, MA: Addison-Wesley.
- Vallacher, R., & Wegner, D. (1987). What do people think they're doing? Action identification and human behavior. *Psychological Review*, 94 (1), 3–15.
- van Rens, L. S. (1997). Usability problem classifier. Unpublished master's thesis, Virginia Tech, Blacksburg, VA.
- Vora, P. R. (1995). Classifying user errors in human-computer interactive tasks. *Common Ground: The Newsletter of Usability Professionals*, 5 (2), 15.
- Wharton, C. (1992). Cognitive walkthroughs: Instructions, forms, and examples (Tech. Report CU-ICS-92-17). Boulder, CO: University of Colorado.
- Wharton, C., Bradford, J., Jeffries, R., & Franzke, M. (1992). Applying cognitive walkthroughs to more complex user interfaces: Experiences, issues, and recommendations. In *Proceedings of the CHI '92 Conference Proceedings*, ACM Press: Monterey, CA, 381–388.
- Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 105–140). New York: John Wiley & Sons.

8 Appendix A. Latent Semantic Analysis

LSA is a machine-learning technology for simulating human understanding of the meaning of words and text. It uses a fully automatic mathematical/statistical technique to extract and infer meaning relations from the contextual usage of words in large collections of natural discourse. It is not a traditional natural language processing or artificial intelligence program; it uses no human constructed ontologies, dictionaries, knowledge bases, semantic networks, grammars, syntactic parsers, or morphologies.

How LSA works. The problem of language learning (for either a machine or a human) can be represented as follows. The meaning of a passage is some function of the meaning of its words:

$$m(\text{passage}_i) \sim f\{m(\text{word}_{i1}), m(\text{word}_{i2}) \dots m(\text{word}_{in})\}.$$

The language learner's problem is to solve a lifetime of such simultaneous equations for the meanings of all the words and thus also the meaning of any passage. To make an approximate solution feasible, LSA makes three strong simplifying assumptions: non-verbal context is ignored, only the textually represented language that a simulated person might have experienced is usually used as data, and the word-meaning combination function is addition. Thus:

$$m(\text{passage}_i) \sim \{m(\text{word}_{i1}) + m(\text{word}_{i2}) + \dots + m(\text{word}_{in})\}$$

A large corpus of text, as similar as possible to the sources from which the humans whose performance is to be simulated would have acquired the knowledge to be simulated, is divided into meaningful passages, such as paragraphs, which are then represented as equations. An approximate solution to the system of equations is found by the matrix algebraic technique of Singular Value Decomposition. (For details see Berry 1992, Berry, Dumais and O'Brien, 1995, Deerwester et al., 1990, Landauer 1998, 1999, Landauer and Dumais, 1997, Dumais, 1991, 1994). Each word in the corpus, and any passage, is represented as a high (typically around 300) dimensional vector. The non-monotonic optimal number of dimensions is important. Dimension reduction constitutes an inductive step by which words are represented by values on a smaller set of abstract features rather than their raw pattern of observed occurrences. One important result of this is that words that never appear in the same document, such as different terms for the same thing or procedure used in different ways or for different purposes or by different organizations, get appropriately represented by similar vectors. Relations between meanings are computed as cosines or other pattern similarity metrics on the reduced-dimensional vectors.

This model yields good simulation of human verbal meaning across a wide spectrum of verbal phenomena and test applications: (1) correct query-document topic similarity judgments, even when there are no literal words in common between query and document (Dumais, 1994); (2) correctly mimicking human word-word semantic relation and category membership judgments (Landauer, Foltz and Laham, 1998); (3) correct choices on vocabulary, and, after training on a textbook, subject-matter multiple choice tests (Landauer, Foltz and Laham, 1998); (4) accurate measurement of conceptual coherence of text and resulting comprehensibility (Foltz, Kintsch and Landauer, 1993, 1998); (5) accurate prediction of expert human holistic ratings and matching of the conceptual content of student essays and textbook sections (Landauer, Foltz and Laham, 1998); (6) correctly mimicking synonym, antonym, singular-plural, past-present, and compound-component word relations, (Landauer, Foltz and Laham, 1998); (7) representing word ambiguity and polysemy, the possession of two or more distinct senses or meanings by the same word; (8) providing significant improvement for language modeling in Automatic Speech Recognition

(Coccaro and Jurafsky, 1998); (9) matching textual work histories to discursive job and task descriptions (Laham, Bennett, and Landauer, 2000); (10) estimating conceptual overlap among large numbers of training courses by analysis of test contents (Laham, et al. 2000); and (11) accurately simulating the phenomenal rate of growth of human vocabulary during K-12 school years (Landauer and Dumais, 1997).

9 Appendix B. The User Action Framework

High reliability in the underlying framework, or agreement among users on how the UAF is understood and used, is a prerequisite to consistent understanding across tool users of how a given usability situation would be interpreted with the tools. After a number of approaches that led to low reliability, as a basis we adopted Norman's stages-of-action model [Norman, 1986], a theory-based model of interaction between human users and machines.

9.1 The Interaction Cycle

As shown in Figure 55, we adapted and extended Norman's model into our *Interaction Cycle*, as an organizing mechanism for the User Action Framework. UAF content is about how interaction design supports and affects users during interaction as they make cognitive, physical, or perceptual actions in each part of the Interaction Cycle.

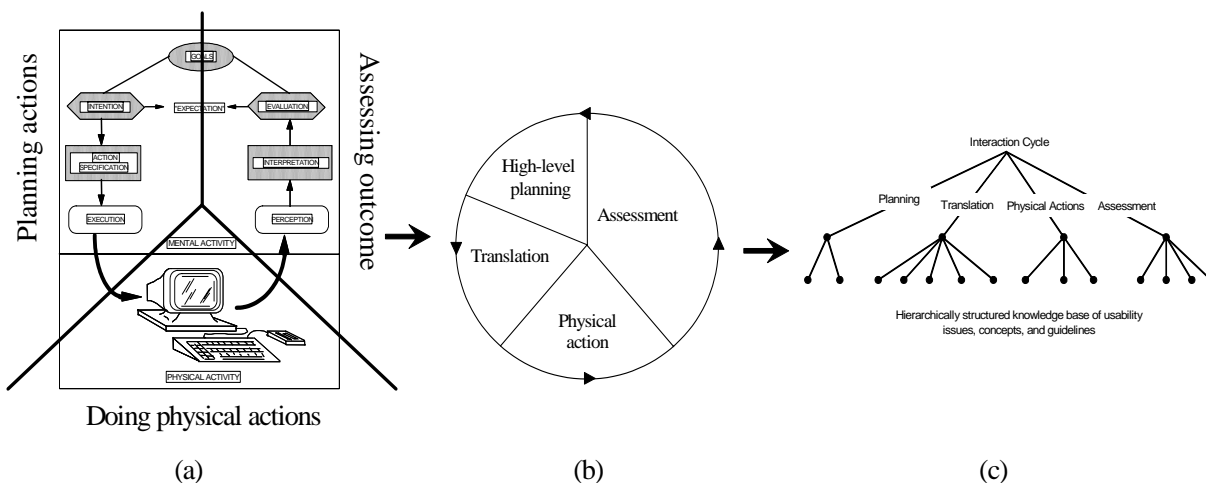


Figure 55: Adapting (a) Norman's stages-of-action model into (b) the Interaction Cycle and combining with (c) a structured usability knowledge base to form the User Action Framework

The Interaction Cycle includes all of Norman's stages, but expands on them and organizes them pragmatically in a slightly different way. Like Norman's model, the Interaction Cycle is a picture of how interaction between a human user and *any* machine happens in terms of sequences of cognitive, physical, and perceptual user actions. This generality served us well, giving us an immediate and significant rise in reliability levels [Andre, Hartson, Belz, & McCreary, 2001] and allowing us to proceed with tool development.

In the general form of our Interaction Cycle, shown in Figure 56, we have preserved Norman's division between cognitive actions (labeled THINK) and physical actions (labeled DO) and have added some additional aspects of perception (labeled SEE) and an indication of the role of outcome within the system and feedback to the user. The terms "think" and "see" are simple terms to connote the broad and complex range of cognitive and perceptual human information

processing that is possible within this cycle. We also divided Norman's planning into high-level planning and translation (explained below).

Planning is the part of the Interaction Cycle that contains all cognitive actions by users to determine *what* to do. Supporting users in planning involves helping them understand the system model and helping them keep track of where they are within a task.

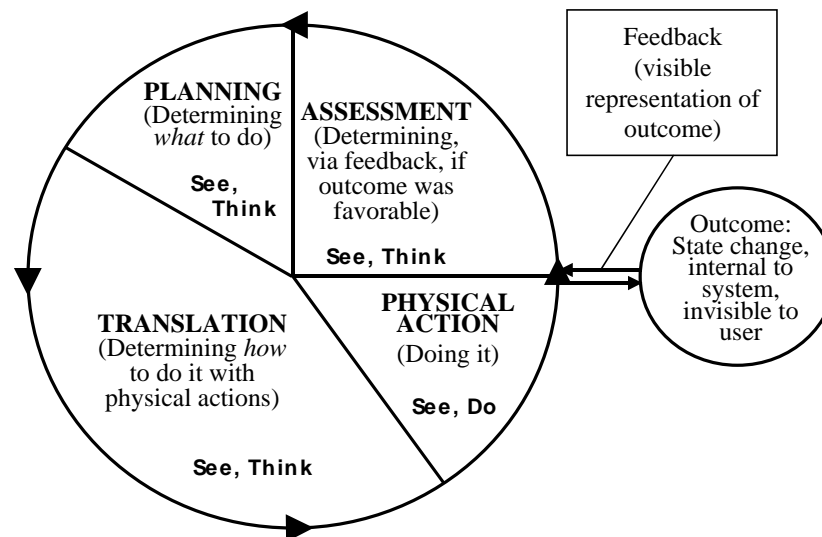


Figure 56: Interaction Cycle of the UAF

Translation is the part of the Interaction Cycle that contains all cognitive actions by users to determine *how* to carry out the intentions that arise in planning. Translation results in an action specification (e.g., click and drag an icon) to accomplish an intention and is the process users employ to cross Norman's Gulf of Execution [1986]. Norman characterizes that gulf as a gap between two languages, the psychological language of the user's work and problem domain and the physical action-object language of the system's physical domain of controls to change the system state. We see the bridging of a gap between these languages as a kind of translation. The most important interaction design element for helping users, especially new users, make this translation are cognitive affordances [Hartson, 2003], features such as visual cues that help the user think or know how to do something.

The **physical action** part of the Interaction Cycle is where users perform physical actions on devices and user interface objects (physical affordances) to manipulate the system (e.g., typing, clicking, and dragging in GUIs; gestures and navigational actions such as walking in virtual environments). For expert users, physical actions become a limiting factor in performance and, therefore, are most important from a design perspective.

A system response, including feedback, is the only representation to the user of outcomes of their actions. **Assessment** is the part of the Interaction Cycle containing all cognitive actions by users to determine, based on system feedback, whether the outcome of planning, translation, and physical actions was favorable, meaning desirable or effective to the user. The assessment part of the Interaction Cycle is about how well the interaction design, and the feedback design in

particular, helps users cross Norman's Gulf of Evaluation [1986] by supporting their ability to determine the success of actions.

As the UAF structure has grown with categories and sub-categories, we have tried to make it as complete as possible. However, absolute completeness is neither feasible nor necessary. The UAF is designed explicitly for long-term extensibility and maintainability.

9.2 The UAF: Adding a Structured Usability Knowledge Base

Figure 13 shows the Interaction Cycle as the top-level organization of the UAF. For several years we have been compiling usability concepts from guidelines, published literature, large amounts of real-world usability data, and our own experience. We organized this information within the Interaction Cycle categories and several levels of sub-categories within the UAF, as a structured, tool-independent knowledge base of usability principles, issues, and concepts. UAF content is expressed in a way that focuses on the analysis of interaction designs and how they support and affect the user during interaction for task performance, as the user makes cognitive, physical, or perceptual actions in each part of the Interaction Cycle.

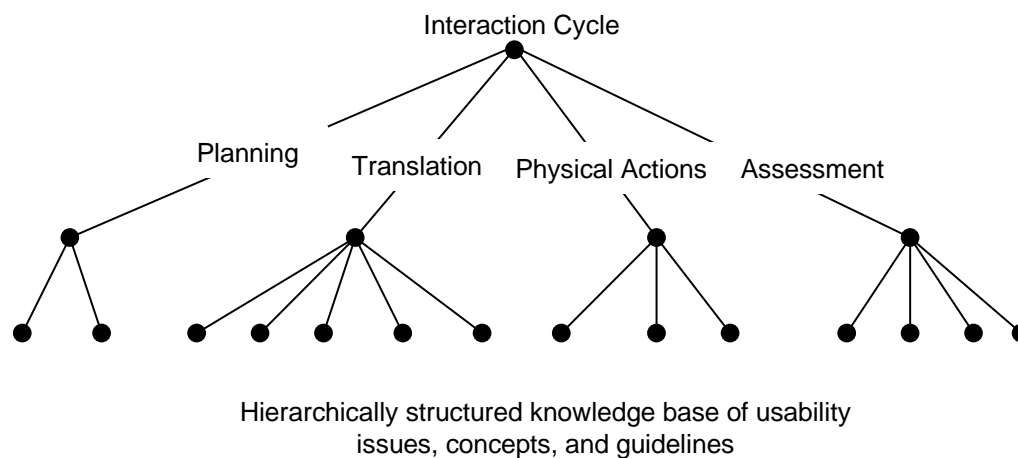


Figure 57: Building the UAF upon the Interaction Cycle

Thus, the UAF derives powerful generality and flexibility in its application across many tools and interaction styles from its theory base in Norman's model. It also derives essential practical credentials from its empirical roots in the real-world usability data and design guidelines on which its content is based.

9.3 Related Work

As early as 1994, a workshop at the annual CHI conference addressed the topic of what to do after usability data is collected [Nayak, Mrazek, & Smith, 1995]. Researchers and practitioners addressed questions about how usability data should be gathered, analyzed, and reported. Without

practical tools grounded in solid HCI theory to support interpretation of usability data, researchers and practitioners were concerned about the reliability and adequacy of usability data in informing the interaction design and redesign process. They wanted effective ways to get from raw observational data to meaningful conclusions about specific flaws and how to remedy them in specific interaction design features.

9.3.1 Model-based frameworks

The User Action Framework is grounded in the cognitive theory base of Norman's stages-of-action model of human-computer interaction [Norman, 1986]. We also give credit to the cognitive walkthrough [Lewis, Polson, Wharton, & Rieman, 1990], which is similar in many ways to Norman's work. Both approaches ask questions about whether the user can determine what to do and how to do it, how easily the user can perform physical actions, and (to a lesser extent in the cognitive walkthrough) how well the user can tell whether these actions were successful in moving toward task completion.

Others have also used Norman's model as a basis for usability inspection, classification, or analysis and found it helpful for diagnosing and communicating usability problems. Cuomo and Bowen [Cuomo & Bowen, 1992] and Cuomo [Cuomo, 1993; Cuomo, 1994] concluded that usability evaluation methods based on a user's complete cycle of interaction, such as the cycle represented in Norman's model, were necessary to capture the full range of usability problems in direct manipulation-style designs. Sutcliffe and his colleagues adapted Norman's model of interaction to a walk-through usability evaluation method for virtual reality interfaces [Kaur, Maiden, & Sutcliffe, 1999; Kaur, Sutcliffe, & Maiden, 1999].

Lim, Benbasat, and Todd [1996] used Norman's model to guide exploration of the effects of interaction style and task familiarity on user performance. They drew on action identification theory [Vallacher & Wegner, 1987] to study the differences in how users follow Norman's model, especially for planning actions, in cases of automated [Bødker, 1989; Bødker, 1991; Logan, 1988] versus self-conscious (controlled) performance.

9.3.2 Classifying usability problems by type

Design guidelines [Mayhew, 1992; Shneiderman, 1998] and heuristics [Nielsen & Molich, 1990] offer a basis for high-level classification of usability problems, but little of this kind of classification is done in practice. In defining heuristics, Nielsen did some classifying and aggregating of usability problems; others have explored classification in general [Mack & Montaniz, 1994], or by source and location in dialogue [Brooks, 1994; Nielsen, 1993]. Simple schemes are used for classifying problems by severity or importance [Desurvire, 1994; Nielsen, 1993; Rubin, 1994], based on type of user error [Vora, 1995], or to aggregate usability problems found by several evaluators [Nielsen & Molich, 1990]. In general, however, these approaches to classification have been ad hoc, incomplete, unstructured, and usually unhelpful for finding solutions to usability problems in an interaction design.

Classification of usability problems by type is not only valuable for practitioners, but is also necessary for researchers who evaluate usability evaluation methods. Studies comparing usability evaluation methods [Cuomo & Bowen, 1992, p. 1255; Desurvire, 1994; Karat, 1994; Muller, Dayton, & Root, 1993] and reviews of those studies [Gray & Salzman, 1998, ca. p. 242; Hartson, Andre, & Williges, 2001] have concluded that some means is needed for classifying usability problems by type, so that strengths and weaknesses of the methods can be assessed in terms of what kinds of problems they are best at identifying.

10 Appendix C. Review of Problems Addressed

10.1 Low Return on Investment Downstream from Usability Testing

For several years, usability has no longer required justification in most quarters: "Usability has become a competitive necessity for the . . . success of software" [Butler, 1996]. Because of the growing awareness of its importance, organizations have been expending resources for "doing usability"—building enviable usability laboratories, buying usability equipment, training developers in usability engineering methods [Hix & Hartson, 1993], and conducting usability testing. However, those same organizations are often not achieving acceptable returns on this investment "downstream" (after testing) in the usability process, where there are few techniques and almost no tools to support problem extraction, analysis, diagnosis, documentation, reporting, and data management of usability problems found during user-based testing.

Having seen many hundreds of usability problem descriptions from real-world usability laboratories (e.g., [Keenan, 1996]), we know that usability problem reports are more often than not ad hoc laundry lists of raw usability problem descriptions reporting what someone believed to be salient, based on what they were thinking at the time. Practitioners need a structured diagnosis framework to establish problem descriptions in terms of "standard" usability attributes for complete and accurate usability problem reporting.

Figure 58 shows a somewhat simplified depiction of the management (identification, analysis, reporting, and fixing) of usability problems within the usability engineering development process. Usability testing at "A" produces raw usability data in the form of critical incident and usability problem descriptions. The problems are later analyzed and diagnosed at "B" and put into a usability problem report database for subsequent fixing with design solutions at "C". After changes are implemented, the process can be iterated with successive testing, analysis, and redesign. A principal problem we address in this proposal is that the analysis and design at "B" usually follows data gathering at "A" after a delay in time, is often done by different people, and frequently occurs at a different location. Similarly, redesign at "C" is often done by different people and after a delay.

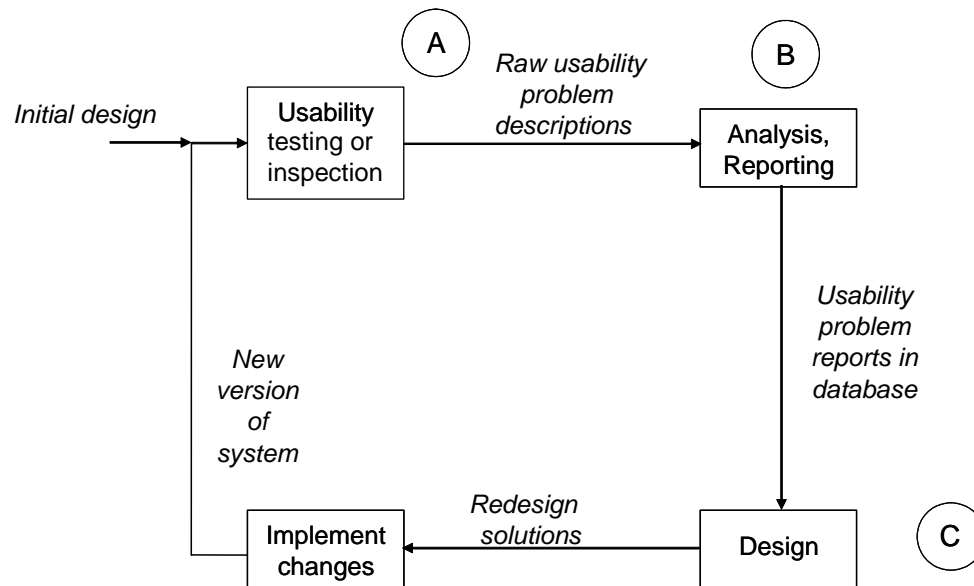


Figure 58. Process for identifying, analyzing, reporting, and fixing usability problems.

The usability practitioner gathering usability data during lab-based usability testing may observe the necessary data at “A” needed later to make an effective diagnosis at “B”. However, our exploratory studies in Phase I have shown that, without guidance from structured tools in systematic data elicitation and collection during user observation, information crucial to later analysis and diagnosis is not captured and, consequently, the value of the expensive usability data declines drastically. Usability problem communication then must rely on human memory and word of mouth, and developers can only try to interpret and reconstruct the missing usability information.

Finally, usability engineering research that *has* been done has been slow to reach practitioners and get into practice. Without an effective bridge from research to commercialization, university developed tools rarely see service in real-world environments.

Our development of a usability engineering framework, combining the technologies of Latent Semantic Analysis (LSA) and the User Action Framework (UAF), and usability engineering support tools offers an alternative vision, a vision yielding return on investment in usability engineering activities downstream from usability testing, by filling the need for:

1. A theory-based conceptual framework organized by usability problem types
2. Integrated usability engineering support tools for
 - a. Systematic and complete usability data collection
 - b. Usability problem extraction, analysis, diagnosis, and reporting
 - c. Usability data management
 - d. Cost effective, theory-driven usability inspection
 - e. Specific, easy-to-apply user interaction design guidelines
3. Extensibility of usability engineering methods and tools to new interaction styles
4. Intelligent systems that can match *ad hoc* wording of observational reports to structured categories and associated diagnostic information
5. Commercial grade web-based tools with appropriate levels of user support.

10.2 Need for theory-based conceptual structure

Gray and Salzman [Gray & Salzman, 1998] noted that, "To the naïve observer it might seem obvious that the field of HCI would have a set of common categories with which to discuss one of its most basic concepts: Usability. We do not. Instead we have a hodgepodge collection of *do-it-yourself* categories and various collections of *rules-of-thumb*." As Gray and Salzman continue, "Developing a common categorization scheme, preferably one grounded in theory, would allow us to compare types of usability problems across different types of software and interfaces." We agree; we believe a conceptual framework and standard usability vocabulary are essential to organize and guide usability problem analysis, diagnosis, reporting, and other post-testing usability development activities.

10.3 Need for integrated suite of usability engineering tools

Since the usability engineering development process is an integrated and interwoven set of activities and a number of unrelated steps, usability engineers need tools to reflect that interconnectedness. We proposed an integrated tool suite centered on a single underlying foundation (LSA knowledge base) and a single supporting database system, in contrast to many separate tools.

10.3.1 Need for theory-based usability problem analysis, diagnosis, and reporting tool

Few, if any, tools exist to support usability problem extraction (isolating individual usability problems from the raw observational usability data), using observational data such as from critical incidents and verbal protocols to isolate and identify specific, separate usability problems. Worse, there are no tools for systematic, structured diagnosis of usability problems for precise, complete problem reporting. Yet usability engineers cannot expect to know what to "fix" in redesign without a complete and precise diagnosis in terms of problem type and subtype and by effect on the user and cause within the interaction design.

Clearly, effective diagnosis is essential to return on investment in usability testing. Usability problems which look similar on the surface can have different underlying causes, and vice versa. To create an effective solution, designers need to understand the problem precisely and to know its causes. Fixing the wrong problem is wasteful and can cause new problems. Not fixing the right problem is a lost opportunity, an opportunity paid for in the usability testing process. Tools like those we have developed can guide the usability analysis process so that, as Gray and Salzman [1998] state, practitioners can use observational data about a problem to implicate a cause in the interaction design.

10.3.1.1 Need for integrated usability data management tool

Practitioners have long identified the need for database support for usability engineering activities (e.g., [Pernice & Butler, 1995]). Very few projects, for example, have usability database facilities to track the life history of each usability problem. There is no project- or organization-wide memory of the process and its results, preventing re-use of usability data and analysis. Practitioners cannot build on lessons learned, nor inventory the types of problems the project is

having, nor see and compare trends within and across projects. We have developed database-supported tools for maintaining, accessing, and visualizing, usability data throughout a project, beyond a single project, and across the developers' organization.

10.3.1.2 *Need for systematic, theory-driven usability inspection tool*

Usability inspection methods have emerged as an economical alternative to lab-based usability testing. However, usability evaluation method studies (e.g., [Andre, Williges, & Hartson, 1999; Jeffries, Miller, Wharton, & Uyeda, 1991; Karat, 1994]) indicate a need to improve current usability evaluation methods in terms of both theoretical foundations and a more focused approach to tailor cost-effective inspection instances. Most existing inspection methods are applied the same way, regardless of the situation. However, if the design is in a very early stage and no details yet exist, developers can save resources by using abstraction to omit questions about detailed design. Similarly, an inspection instance on behalf of expert users would be more efficient if it were focused by omitting questions about supporting inexperienced users.

10.3.1.3 *Need for usability design tool to guide interaction design and redesign activities*

In an informal survey among Web designers at a prominent corporation (name withheld by request) to explore the use of design guidelines, we learned that:

- most designers don't use guidelines;
- those who do use them, find them difficult to locate and difficult to apply;
- most designers feel they don't get the benefits they should from existing guidelines; and
- most designers feel that guidelines are in a form that is too general or too vague to apply to specific design problems.

Most developers agreed that effective use of guidelines could and should lead to better usability in designs. However, existing design guidelines are scattered across a large body of on-line and off-line professional and academic literature. The process of finding appropriate guidelines, interpreting them in terms of a specific design situation, and applying them to determine a design solution is a process that is both labor-intensive and not well understood. Developers need a tool to allow the average practitioner easily to apply guidelines to a specific usability design situation.

10.3.1.4 *Need for extensibility to new interaction styles*

Most existing user interaction development methods are limited to graphical user interfaces (GUIs), and now web-based applications. However, the world is rapidly expanding to additional new interaction styles, such as those found in virtual environments, pen-based interaction, and voice interaction, to which GUI-specific guidelines, methods, and tools are not always directly applicable. Practitioners need extensible methods and tools to address newer interaction styles, even styles not yet imagined. The UAF has been designed to be both general and extensible to accommodate these kinds of new interaction styles, and we have made significant extensions to it as part of the work reported here.

10.3.1.5 *Need for intelligent analysis systems to match solutions to usability problems*

The language used to express observations in usability problem reports rarely if ever resembles the language used to classify and diagnose such problems in the usability literature. Typical word matching text analysis systems are therefore insufficient to analyze the ad hoc lab reports in order

to identify the underlying problem type as classified in a theoretical framework. A method for text analysis based on **key concepts, not keywords**, is needed for the desired usability engineering tools to succeed.

11 Appendix D. Benchmark Tasks and Usability Problems from the Formative Evaluation of the DCART Low-Fidelity Prototype

11.1 Example Benchmark Tasks

1. Project “Click” will need a new trial within version 0.1 and iteration “I1”. Please add a trial that begins on Feb 5th 2005 and will be named “T3”. The finishing date is unknown.
2. Create a new benchmark task for Project “Click” and give it a name you want. The user class should be developer. The task should say “Retrieve user data on Click’s administrator page.”
3. Create a project for the VT organization. The new project is called “Debugger” and its purpose is to examine student code for errors. The manager for the project will be Tim Russel and the start date for the project will be Feb 12th 2005.
4. There is a typographic error on Project “Click” description, it should say “Click is a web application” not “Click is is a web-based web application”. Please make the necessary corrections.
5. Add two facilitators to project “Click”. Their names are Anilu Kooper and Manuel Mickenly.
6. Add a facilitator to project “Click” that does not already exist.
7. A new user class must be added to the project “Click”. The user will be a Customer, and they are the ones that will be interacting with the web application.
8. The Administrator user class description is a bit out of date. The administrator’s responsibilities are to monitor the system, maintain database integrity, add new features, and troubleshoot problems on the Click system. After typing these changes you decide not to do it now, so please do not confirm any changes.
9. Create a new benchmark task for project “Click”. The task will ask a customer to create a new user account for the Click web application to allow access to tools within the site.
10. A new usability specification called “Initial Learnability” needs to be created in the project “Click”. In this specification we are checking for how well users with no experience with the “Click” system will perform on initial use. We will measure this by having the users do the task “Create registration overview”. The performance will be evaluated based on time to complete task. As of right now 3 minutes will be our initial value but our aim is to have it done in 2 minutes.
11. A new and bright idea just hit you. You would like see how well the same participant does in a second similar task to benchmark# 10. In particular, you wonder if the performance of new users can improve after just one visit to Click’s web site.

You want to create a usability specification similar to an existing one called “Accuracy – Errors”. The benchmark task for this usability specification should be “Create Registration Form”. We want to evaluate the user on how much they have learned and remembered from the previous task.

Again the performance will be evaluated on the number of errors, as of right now 2 errors will be a good initial value, but 1 error is we are aiming for.

12. A new trial needs to be created for project “Click” within version 0.1, and iteration I1. This trial will be named “T3” and will beginning today and has no known end date. The trial should have only Jon Howarth and Susan Isreel as facilitators. The participants should only be Sarah McGonnal, and “Learnability” should be the only usability specifications associated with this trial. Information will be gathered with the help of Morae¹⁴.
13. We need a new trial for project “Click” to further test more features. However our test will be similar to that of Trial T1. The benchmark task and usability specification will be the same but the Facilitators will only have Steve Jobs. We will also have Sarah McGonnal and Stephenie Raider as participants.

11.2 Usability Problems

1. Incorrect date format
2. Save button expected to be at bottom
3. Inconsistent placement of save and cancel button. Sometimes they are on the top or on the left side.
4. Unable to recognize edit icons due to poor cognitive affordance
5. Unable to recognize add icons due to poor cognitive affordance
6. Facilitator links not descriptive enough to indicate subset of possible actions
7. User class link not descriptive enough to indicate subset of possible actions
8. Project Team link not descriptive enough to indicate that it has team members: managers, facilitators
9. Expecting the word “user class” to be more specific such as “end user class” to indicate proper meaning since DCART has multiple hierarchies of users within its system.
10. Side tracking task does not bring user back to where they left off. User is confused and lost afterwards and backtracking is required to recover
11. Inconsistent layout of project team and user class since both use the same table structure layout but content within conveys different methods of how to view / modify information
12. Inconsistent layout of links on top menu and use of icons/button links

¹⁴ See <http://www.techsmith.com/morae.asp>.

13. Domain specific vocabulary not explained (e.g. Measuring Instrument, Usability Attribute, or Baseline Value)
14. Missing units for baseline value or target value to indicate number of errors or time in seconds, minutes, or hours
15. No direct way to add facilitator from trial level
16. User does not notice that the back link even exists
17. Hierarchical structure of project facilitator and facilitator in organization not understood
18. Inconsistent usage of icon vs links, user expects an icon but sees a link and vice versa
19. Method of trial duplication not consistent with other duplication process
20. User does not understand the hierarchical structure of facilitator in organization and facilitator in project.

12 Appendix E. Exploratory Tasks for Participants in the Field Test of the DCART High-Fidelity Prototype

The suggested exploratory tasks are to get you started; you should definitely not be limited by them. We would especially like you to explore the tool and try to do other things that you would normally do in your own usability development environment. We value feedback specific to your own, or your organization's, business model and approach to doing usability engineering.

Part A: Entering administrative definitions

Before you start, we need to establish some simple terminology for describing tasks for you to try in your evaluation of DCART. There is potential for confusion, since you are doing an evaluation of a tool that is used to support usability evaluation. To avoid confusion and clearly distinguish the two kinds of evaluation involved, we will assume that everything you do here is toward the goal of DCART evaluation. All the tasks you will try with DCART will be oriented-toward usability evaluation of some other project, prototype, or product, which we will refer to as the “target project”. As a target project in these tasks, you can use one of your own recent projects, some other project with which you have some experience, or one that is made up to simulate or represent the kind of project in which you might use DCART to support usability evaluation.

1. Create a new organization, giving it your own organization's name and organization information.
2. Create a new (target) project, giving a name and project information of your own choice.
3. Create a new version, giving a name and version information of your own choice.
4. Create a new evaluation session for this version, naming it S1 (or any other name of your choice). Give any other session information of your choice. Create a new participant for this session, using the “just-in-time” sideways link in the session record.
5. Create a new user class for your target project, giving it an appropriate name and filling in the user class characteristics.
6. Create a new benchmark task to be used to in usability evaluation of your target project. Make this first benchmark task a fairly simple, representative task for your target system users.
7. Create a new task run record under the current session (the one you just created above), giving it a name that reflects what you will be trying to use it to evaluate in your target system. Select the user class and benchmark task that you have just created.
8. Use the sideways link in the task run record to go off and define a usability specification for this task run. Choose an appropriate usability goal from those available in DCART or go back and create a new one and use it in this usability specification. Do the same for a usability attribute, the measuring instrument, and all the other parts of the usability specification. Make sure you set it up to involve some quantitative data, such as time on task and/or error counts.

9. For further practice and usage evaluation, define additional projects, versions, sessions, task runs, user classes, benchmark tasks, usability specifications, etc.
10. Create a new usability specification by reproducing an existing one and adapting/modifying it.

Part B: Usability data collection and problem diagnosis

As with the previous evaluation tasks, the usability data collection and problem diagnosis tasks must be performed in the context of a real, imagined, or simulated project development environment. You will obtain the most realistic and ecological DCART evaluation by using it to do real usability evaluation within a real target project. If this isn't possible at this time in your organization, please do your best to simulate the most realistic project usability evaluation situation that you can.

IMPORTANT: If you are not doing a real usability test while evaluating the data collection aspects of DCART here, it is essential that you have a few “real” (as realistic as possible) usability problems ready to include in these tasks, as usability problems identified within your simulated target project system (the one you are using DCART to help you evaluate).

1. Navigate to the task run that you created previously for the “Part A: Entering administrative definitions”.
2. Review the task run setup, the user class and user class definitions, and the usability specification you created.
3. Go to the collect and review mode of the task run and get prepared to do some usability data collection on your real or simulated project, with your first participant on the first benchmark task.
4. Start your “participant” doing the task and simulate taking quantitative data (e.g., time on task, error counts) and qualitative data (e.g., initial usability problem records based on observed critical incidents, verbal protocol from the participant, etc.). This usability evaluation activity is one of the busiest for the usability practitioner and one place where the best support by DCART is essential. Thus, this aspect of DCART usage is among the most critical and requires lots of attention from you in your DCART evaluation.
5. For each real or simulated usability problem (from your pre-prepared list), create a new usability problem record within the task run using DCART and do as much of what is required as you can, including making a guess as to the Immediate Intention in terms of location within the UAF Interaction Cycle.
6. When your participant has completed the task performance, close the data collection form and follow the link (in the Task Run record) to the usability problem review form while your participant is still present. Quickly review the project context and summary of quantitative measurements. Review the initial problem description. Ask questions of the participant (simulating this, as necessary) to get at the essence of the problem, including the user's immediate intention, using the Wizard to confirm or change your choice for the location within the Interaction Cycle, as part of the immediate intention.
7. Save your problem records and, at the end, close the data collection form and the task run.

8. Pretend that some time has passed and you wish to review and further analyze and document the usability problems found in this task run. In the left-hand side of DCART, navigate to this task run and select usability problems to look at all the problems associated with this task run.
9. Select one specific usability problem record and open it. Go through all the fields and, using your imagination, fill in realistic information about the problem. Refine the problem name and description, as needed. In the designer knowledge field, put in something that you, as a developer, might know about the problem, but the user/participant might not, something to help reveal the real nature of the problem from the design viewpoint.
10. Next you will be diagnosing this usability problem, pin-pointing the exact problem in the context of its causes within the interaction design, so that the correct problem gets reported and fixed. This is one of the more complex activities in using DCART, representing an important step not offered in other usability-engineering-support tools. If necessary, please review the tutorial “Tutorial 4 - Usability Problem Diagnosis”, so that you get the most from this part of the evaluation using DCART.
11. Scroll down to the diagnosis field and use the UAF to determine a diagnosis for this usability problem by finding the UAF path that best describes the problem type and cause within the interaction design. Return from the UAF to the problem record, bringing the diagnosis information.
12. In our experience, it takes some practice to feel comfortable with the problem diagnosis process, so please repeat the above diagnosis step for several other usability problems, making up new problems or taking them from your own experience as necessary, until you feel you are familiar with the process.

Table of Figures

Figure 1: Screenshot of the DCART component of the Software Therapist system.	18
Figure 2. The ST Browser opened to the UAF, showing the contents of the root node of the UAF.	18
Figure 3: Levels of hierarchical context and associated resources in DCART.....	18
Figure 4: User Action Framework as a tree structure.....	18
Figure 5. Wizard decision structure.....	18
Figure 6. Support for levels of context and associated resources.	18
Figure 7. Workspace view.....	18
Figure 8. Edited expanding list record.....	18
Figure 9. Modification options bar and selection checkbox.....	18
Figure 10. Session record.	18
Figure 11. Task run record.	18
Figure 12. Usability problem collection form.	18
Figure 13. Usability problem review form.	18
Figure 14. Data view.	18
Figure 15. usability problem record.	18
Figure 16. UAF Diagnosis form.....	18
Figure 17. Node detail view.	18
Figure 18. The Diagnosis Wizard.....	18
Figure 19. The Software Therapist (ST) Browser is launched from a web page.....	18
Figure 20. A user management system can control access to different resources, such as integrated electronic copies of articles and textbooks.	18
Figure 21. Software Therapist (ST) Browser, showing the Library View.....	18
Figure 22. Help pages provide a guide to the ST Browser's features and how to use them effectively.....	18
Figure 23. Detail of the TOC view.....	18
Figure 24. ST Browser showing the selected node highlighted in the TOC, the path to the current node shown in the Path Summary, and the node's content (title, description, examples, and cross-references) shown in the Main Display area.	18
Figure 25. Navigation by browsing history.	18
Figure 26. Navigation using the Path Summary panel.	18
Figure 27. Selecting a Problem Report Set.....	18
Figure 28. Selecting a Problem Report.....	18
Figure 29. Invoking a UAF Diagnosis search on the text of a problem report.	18
Figure 30. Ranked results displayed in the Search Results panel, showing a 42% relevance ranking for the top search result, a UAF diagnosis path terminating at the UAF node "Correct expression of meaning..." 18	
Figure 31 UAF Diagnosis search results showing the top search result in the TOC, the path summary view, and its contents in the main display panel.	18
Figure 32. Ranked search results with rollover tool tip showing degree of relevance/similarity and overview of result diagnosis path.	18
Figure 33. The 6 th -best search result.....	18
Figure 34. Using the Top Query Results pop-up box to navigate to another search result.....	18
Figure 35. Limiting search scope for follow-up search.	18
Figure 36. Results of the scoped search.....	18
Figure 37. Entering the text for a new problem description into the query box, then invoking a UAF Diagnosis search.....	18
Figure 38. The ST Browser after the user has selected the 7 th -best result of a UAF Diagnosis search on the new query text.	18
Figure 39. Invoking a Problem Report Search from the UAF.....	18
Figure 40. The results of the Problem Report Search invoked from the UAF.....	18
Figure 41. Browsing the Problem Reports in the ST Browser.	18
Figure 42. Searching an online textbook for semantically similar sections using Document Search.....	18

Figure 43. Search results.	18
Figure 44. Invoking Document Search on a portion of text selected with the mouse.	18
Figure 45. Invoking a UAF Diagnosis Search on text from a related document, pasted into the query box.	18
Figure 46. Results of UAF Diagnosis Search on text from a related document, pasted into the query box.	18
Figure 47. Results of a Document Search on the same text in another related-literature resource. Note that the degree of relevance (LSA-based similarity) is not simply a matter of the most (or greatest density of) keyword hits, as suggested in the (LSA-based) ranking of search results: The top-ranked result has fewer keyword hits than several other sections.	18
Figure 48. Using the pop-up context menu in the main display panel to show pre-computed links to the 10 most similar nodes in the UAF.	18
Figure 49. The UAF after the user has navigated to the node titled “Font Size” under the main <i>Translation</i> branch of the UAF.	18
Figure 50. Invoking the <i>Search Selected Text</i> function from the main display of the ST Browser.	18
Figure 51. Result of the <i>Search Selected Text</i> function. The text selected as the query in Figure 50 is copied automatically to the query box (bottom left), the TOC is opened to the top search result and indicates the degree of similarity to the query, and the content of the top search results (UAF node 231) is loaded into the main display panel. Matching keywords from the query are highlighted.	18
Figure 52. ST Browser showing the 2nd best search result after a “Document Search” for UAF nodes using arbitrary text pasted into and/or edited in the search query box (at bottom left).	18
Figure 53. Screenshot of some example usability problems from the UAF usability database.	18
Figure 54: UAF Diagnosis form.	18
Figure 55: Adapting (a) Norman's stages-of-action model into (b) the Interaction Cycle and combining with (c) a structured usability knowledge base to form the User Action Framework	18
Figure 56: Interaction Cycle of the UAF.	18
Figure 57: Building the UAF upon the Interaction Cycle.	18
Figure 58. Process for identifying, analyzing, reporting, and fixing usability problems.	18